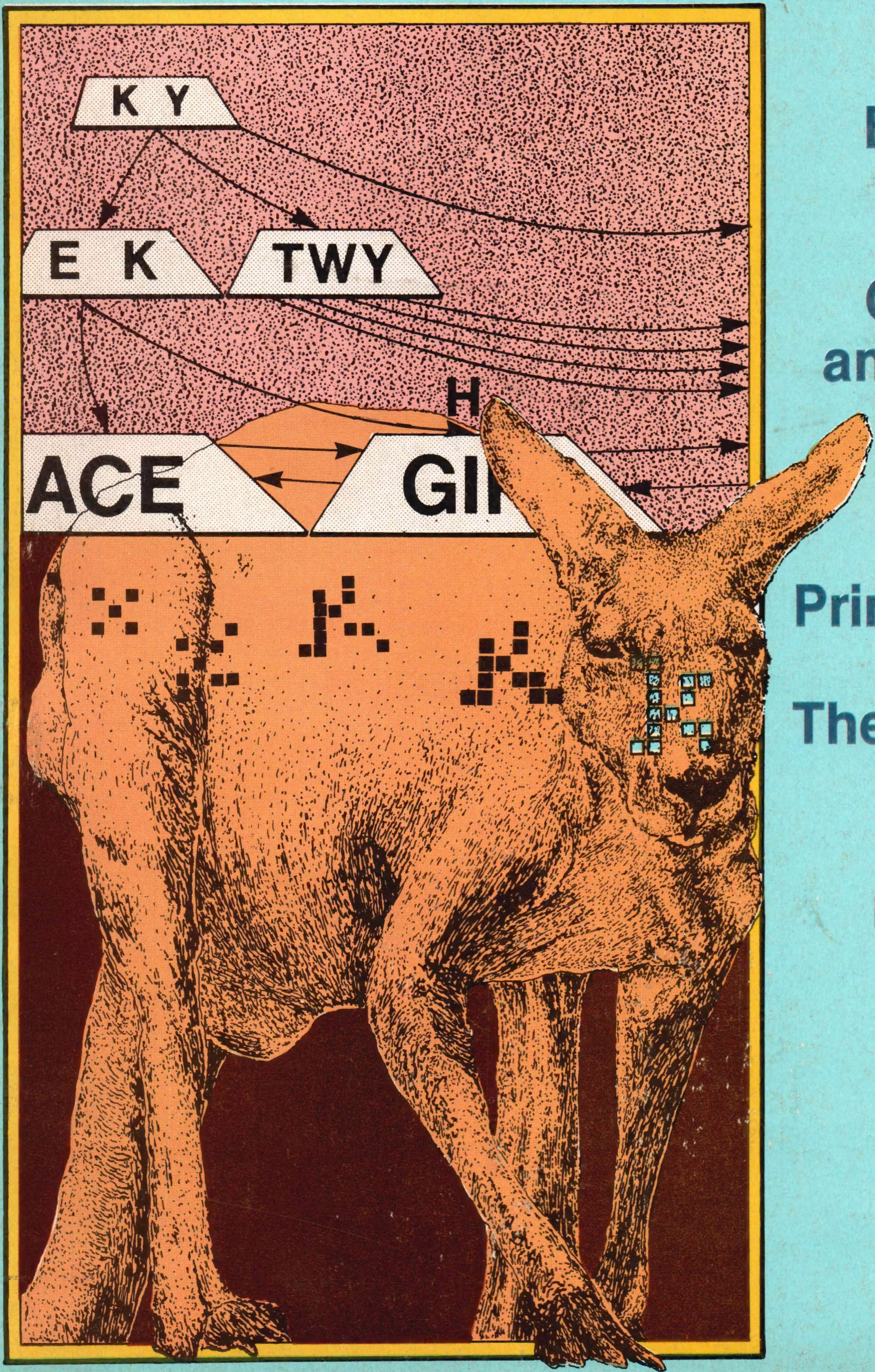
#80 June 1983 --10-84 $200 50 ($3.00 in Canada)

Dr. Dobb's Journal

For Users of Small Computer Systems



B-Tree ISAM Concepts

CP/M BDOS and BIOS Calls for C

IBM PC: Printing Graphics and The Game of Life

. . . and much more!

Dr. Dobb's Journal
For Users of Small Computer Systems

June 1983 Volume 8, Issue 6

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: DeWitt S. Brown, Burks A. Smith, Robert M. Connors, Robert C. Luckey, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Thomas Davis, Jan Steinman, G. Hunter, Ronald E. Johnson, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr., Ted A. Reuss III, Infoworld, Stan Veit, Western Material Control, S.P. Kennedy, Ed Moran, W.D. Rausch. **Lifetime Subscriber:** Michael S. Zick.

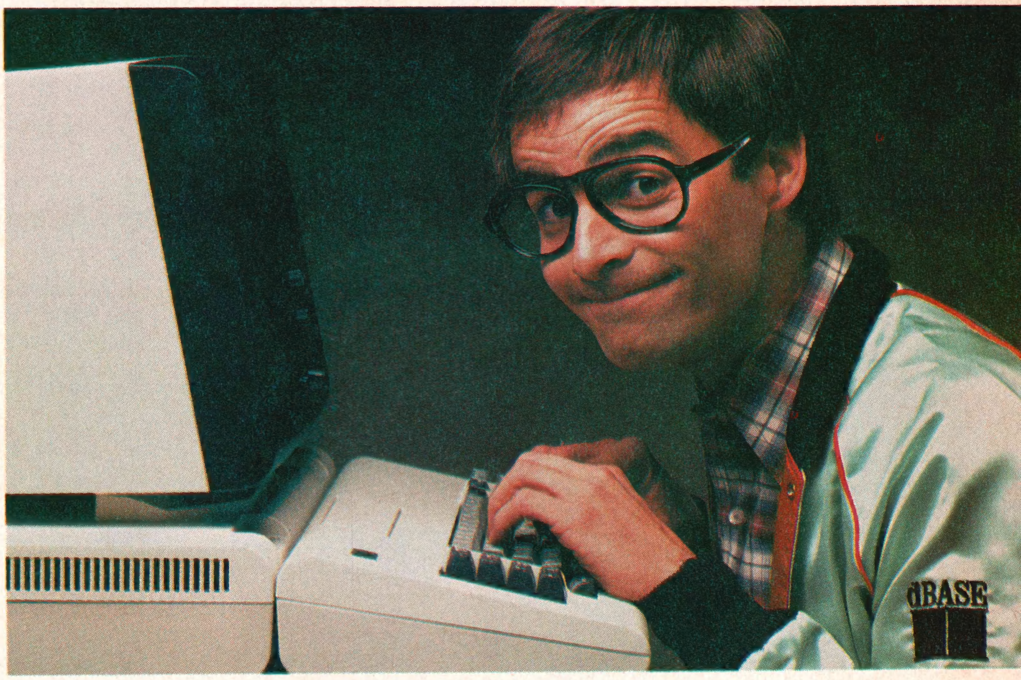
Foreign Distributors UK & Europe: Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014 Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian distributors)*

CONTENTS

ARTICLES

DEPARTMENTS

All you dBASE II™ hotshots are about to get what you deserve.



You've written all those slick dBASE II programs.

Business and personal programs. Scientific and educational applications. Packages for just about every conceivable information handling need.

And everybody who sees them loves them because they're so powerful, friendly and easy to use.

But that's just not good enough.

Uh-uh.

Because now you can get the gold and the glory that you really deserve.

Here's how.

We've just released our dBASE II RunTime™ application development module.

And it can turn you into an instant software publisher.

The RunTime module condenses and encodes your source files, protecting your special insights and techniques, so you can sell your code without giving the show away.

RunTime also protects your margins and improves your price position in the marketplace. If your client has dBASE II, all he needs is your encoded application. If not, all you need to install your application is the much less expensive RunTime module.

We'll tell the world.

With your license for the dBASE II RunTime module, we provide labels that identify your program as a dBASE II application, and you get the benefit of all the dBASE II marketing efforts.

We'll also provide additional "how to" information to get you off and running as a software publisher sooner.

And we'll make your products part of our Marketing Referral Service. Besides putting you on our referral hotline, we'll publish your program descriptions and contact information in *dBASE II Applied*, a directory now in computer stores world-wide.

Go for it.

But we can't do any of this until we hear from you.

For details, write RunTime Applications Development, Ashton-Tate, 10150 West Jefferson Boulevard, Culver City, CA 90230.

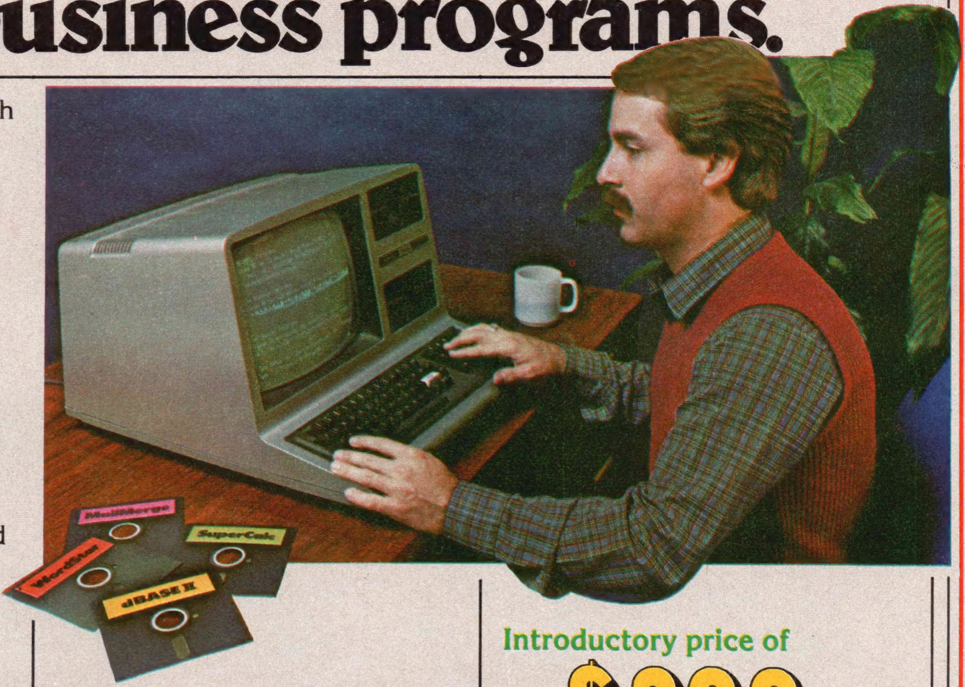
Or better yet, just call (213) 204-5570. And get what you deserve today.

dBASE™ II

ASHTON·TATE

Circle **no. 5** on reader service card.

Put 64K CP/M® 2.2 in your TRS-80 Model III and tap into 2,000 business programs.



Now you can run programs such as WordStar, dBASE II, SuperCalc, MailMerge and virtually thousands of other CP/M-based programs on your TRS-80 Model III.

CP/M 2.2 is the industry standard operating system that gives you access right now to over 2,000 off-the-shelf business programs.

Our plug-in Shuffleboard III comes with 16K of RAM, giving your Model III the power of full 64K CP/M 2.2 without interference of the ROM or video memory. In fact, the Shuffleboard will appear transparent in the TRS-80 mode and will not interfere with any DOS operation.

READ and WRITE Osborne, Xerox and IBM personal computer software plus many more popular formats.

Unfortunately, there is no standardized CP/M format for 5¼" diskettes. But we have developed a way to READ/WRITE and RUN standard programs under the following single-sided formats: Osborne 1 S/D, Xerox 820 S/D, IBM PC* D/D for CP/M 86 only, Superbrain D/D, Kapro II D/D, HP 125 D/D and TeleVideo D/D.

*Will Read and Write Only.

Easy plug-in installation.

It's so simple. The Shuffleboard III plugs into two existing sockets inside your Model III. There are no permanent modifications, no cut traces and no soldering. You'll be up and running in minutes.

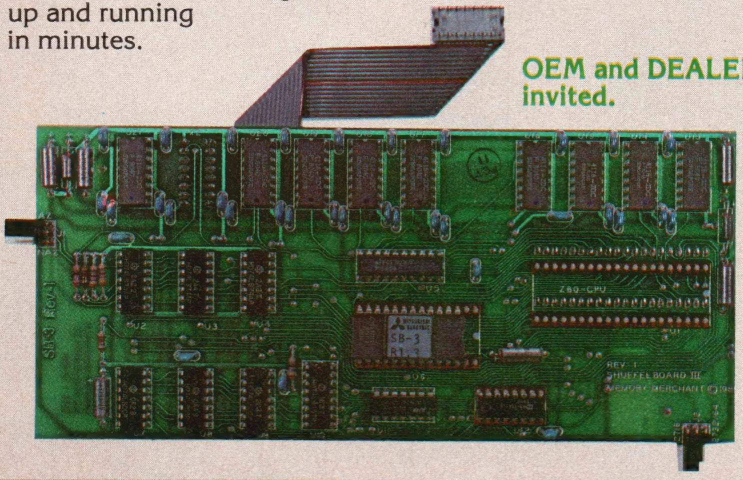
New Products.

80 × 24 VIDEO BOARD: Features dual intensity screen, programmable cursor control for block, underline & blink rate, on-board bell with audible keyclick, battery-operated real time calendar/clock, full ASCII character set plus 256 special character graphics, dual RS-232 outputs and composite video output.

FLOPPY DISK CONTROLLER: Now you can access 5¼" and 8" floppy disk drives in any combination up to 4 drives of S/D density, S/D sided. Tap into a wealth of CP/M software which comes on 8" IBM 3740 format or Pickles & Trout CP/M for the Model II.

SOFTWARE: Additional CP/M software programs are available. Call or write for details.

OEM and DEALER inquiries invited.



Introductory price of

$299.

The Shuffleboard III comes fully burned-in and tested complete with 64K CP/M 2.2 and MBASIC 80 interpreter, plus software manuals and a first class user's manual — with a 1-year limited warranty and 15-day no-risk free trial — for only $299.

See the Shuffleboard III at your dealer's now.

Once you see what the Shuffleboard can do for your Model III you'll want one at once. If your dealer does not yet stock the Shuffleboard have him give us a call. Or send check, money order, VISA or MASTERCARD number (sorry, no COD's) plus $5 shipping per board ($17 outside the USA & Canada)* directly to the address below. Cal. residents please add sales tax. Credit card purchases can be phoned in directly and we'll ship from stock.

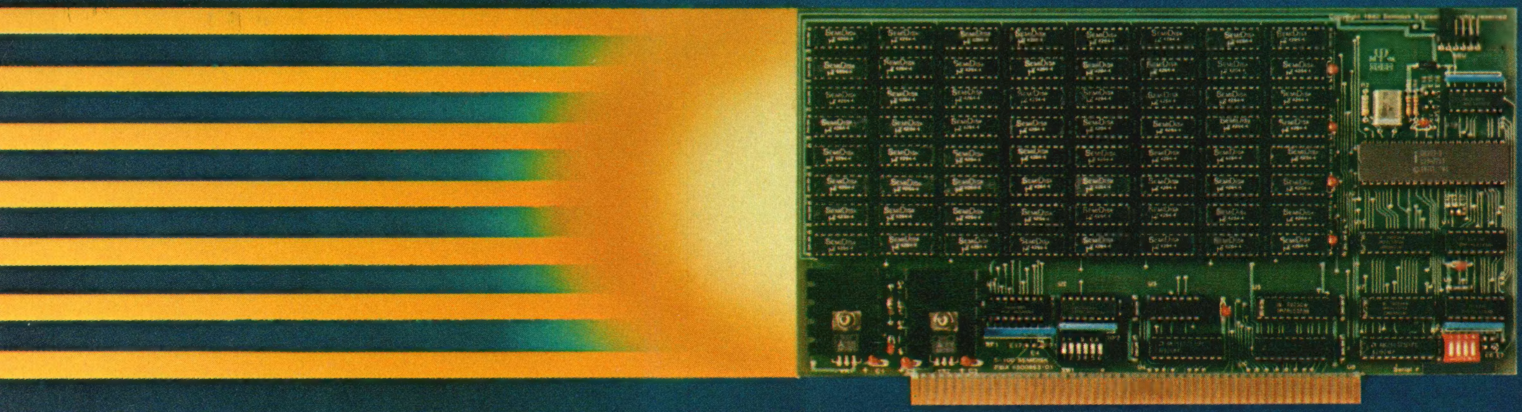
(415) 483-1008

*Air mail shipments to Canada & all other countries.

Memory Merchant™
14666 Doolittle Drive San Leandro, CA 94577
(415) 483-1008

Circle no. 51 on reader service card.

THE PRICE OF *FAST* WAS JUST SHATTERED!



256Kbyte SemiDisk™ $995

For more than a year, we've been making the most advanced disk emulator available for microcomputers. The one that's taken the "waiting" out of computing. Now, we have some more news that'll set the world on fire: A price cut! The NEW 256Kbyte board is only $995. And the 512Kbyte SemiDisks for the S-100 and TRS-80 Model II are $1495. (1Mbyte unit is $2350.) So, what are you waiting for?

The SemiDisk is the **ORIGINAL** single-board microcomputer disk emulator. It has a greater storage density than any other: 1 Mbyte per board! And we've been shipping them for over a year! We didn't do this with 'me too' engineering. Our products are true innovations, based on reliable technology and proven designs, without the need for custom components.

Floppies are ok for data transfer or long-term storage. But they fall far short as online storage. If you are using high level languages, spelling checkers, word processors, databases and other disk-intensive software, you know the price you are paying: time. Your productivity is going down the drain. The SemiDisk disk emulator will save time and increase your productivity.

Even better, Release 5.0 of the SemiDisk CP/M-80 installation software contains SemiSpool, an automatic printer buffer. No extra hardware is required; it's all in the software. Up to 8 Mbytes of buffer space! It's a better solution than a $350 64Kbyte printer buffer that wastes space on your desk. Send documents of almost any length to the printer at a very high speed, then continue using the computer immediately. No Waiting!

SemiDisk

It's the disk the others are still trying to copy.

SemiDisk Systems, Inc.

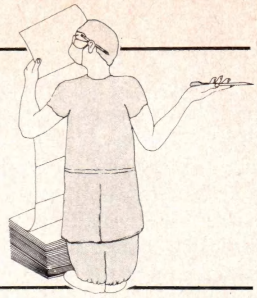
P.O. Box GG Beaverton, OR 97075 (503) 642-3100

Call 503-646-5510 for CBBS®/NW, a SemiDisk-equipped computer bulletin board.

Circle **no. 73** on reader service card.

NO WAITING

DR. DOBB'S CLINIC



by D. E. Cortesi, Resident Intern

A Yodel from Yap

John Thayer Jensen, of Yap, TT, writes us as follows. "I have the kiss of death: I bought an IMSAI VDP-80 and that put IMSAI out of business. So the first thing is, if there are any IMSAI VDP-80 owners out there who will write me, I can share a partially commented disassembly of IMDOS's BDOS, similar disassemblies of the DIO and VIO ROMs, and can sometimes answer questions about IMDOS and IMSAI hardware.

"I also bought a now-obsolete multi-tasking operating system called FAMOS from MVT Microcomputing. They never sold any more (as far as I can tell) and while they make a token effort at support, they can't be expected to do much – and the thing doesn't work. If there are any FAMOS users reading this, I have next to nothing to offer, but an infinite number of questions to ask someone!

"It would be wonderful to hear from some fellow VDP-80 and/or FAMOS users. I can't check into CBBSs, can't go to meetings, can't phone anyone. We have a U.S. zipcode, proving that we are not at the end of the world (although on a clear day, you can see it from here), and domestic first-class mail will reach us by air."

If you would like to correspond with Jensen in his lonely outpost (Yap is an island in the mid-south Pacific), you can write to him at P.O. Box 358, Yap, TT 96943.

PC Still Backward

In March we expressed our puzzlement at the results of printing an ASCII backspace from BASIC in the IBM PC. The result of

PRINT CHR$(8)

is precisely nothing, while the result of

PRINT "wotizzit:",CHR$(8)

is (as we described it) a small reverse-video diamond which we couldn't find in any manual.

Several readers responded. Randolph Fritz of Mahwah, New Jersey and Bob Taylor of Buffalo, New York wrote to point out that the complete character set is shown on pages C-12 and C-13 of the *IBM Technical Reference,* and it includes the one we couldn't find. Robert Pirko of New York added that "The funny character you see is intended to be a reverse-video *circle*. For the graphics adapter in the 40-character mode, it looks roughly like a circle. In the 80-character mode, the width is cut in half and the circle does look more like a diamond."

David Kellogg, also of New York, called to say the same, and added that if you print character 226, the *Technical Reference* says you should get a lower-case gamma, but you actually get an upper case gamma.

OK. The whole situation is very confusing, but we think we understand it now. Follow us through this. (1) IBM, for whatever reason, implemented a graphic symbol for every byte value from 001 to 254 inclusive, leaving only 000, 032, and 255 as blank or null displays. The bytes from 032 to 127 have their ASCII symbols; the others produce unique special characters. (2) The PC's ROM implements a simple dumb-terminal function (WRITE-TTY) which takes ASCII bytes and displays them. It implements the control characters as a teletype would, including carriage return (CR), line feed (LF), and backspace (BSP). However, WRITE-TTY does *not* implement cursor addressing (as any modern terminal should, no matter how dumb).

But (3) the BASIC interpreter needs direct cursor addressing for its screen editing, so it calls upon the more primitive routines in ROM. These routines supply cursor positioning and screen output (SET-CPOS and WRITE-AC-CURRENT). Since they are not trying to emulate a terminal, they treat ASCII control characters as data. A control character like BSP, sent via WRITE-AC-CURRENT, will go into the screen display as a symbol; it will not have its standard ASCII effect.

Then (4) the BASIC interpreter also used the primitive ROM functions to implement the PRINT statement; thus any character written via PRINT will go straight to the screen as a symbol.

Except that (5) the interpreter wants to supply some modicum of standard control characters, so it intercepts BEL (and toots the horn), CR (and returns the cursor), TAB (and tabs by 8), LF (and moves the cursor down), and FF (and clears the screen). In effect, it does the job of WRITE-TTY, but in the interpreter code. It *should* intercept BSP as well; it apparently does so when the cursor is at the left margin, but not otherwise. So a backspace at the left margin acts like a backspace (doing nothing), but once away from the margin, it gets through and becomes a symbol. This appears to be a bug in BASIC.

Finally (6) the interpreter wants users to be able to move the cursor freely, so not only does it implement the LOCATE verb, it *also* intercepts the ASCII control characters US, RS, FS, and GS and treats them as commands to move the cursor in the four cardinal directions.

The net: a BASIC programmer has *two* ways to position the cursor (neither compatible with any other BASIC), but lacks both a standard backspace and the nonstandard symbols of bytes 028-031. Meanwhile, the user of another high-level language can use the backspace (since most of these use WRITE-TTY) but cannot move the cursor. This is how your major corporations do whatcha call your systems analysis and design stuff, see?

SuperKludge

Nick Hammond, now of Torrens, Australia, has sent us a lovely kludge for CP/M 2.2. Here's what he says.

"A couple of months ago, I was asked to modify an 8-inch, single-sided, single-density CP/M system to allow a larger-than-standard directory. It proved to be a fairly simple mod and I thought you might like to share it with your readers.

"A standard 8-inch, single-density CP/M diskette has a file directory that will hold 64 entries. Since each 16K extent of a file requires one entry, this may amount to less than 64 actual files. Given the maximum data storage of 240K odd, this is a reasonable number, but occasions will arise when we need more. The example that prompted this note was a need to fit a dBase II application with a large number of command files onto one disk, leaving the second free for data.

"Fortunately, CP/M 2.2 is both flexible and well thought-out, and expanding the directory can be done relatively simply. Two things must be done: generation of a diskette which will look okay to both the standard and modified systems, and insertion of a two-byte patch to modify the BIOS for the new directory size.

"The standard directory on a single-density 8-inch disk occupies the first two 1K blocks after the system tracks. The modified directory will occupy the first four, and we therefore need to reserve the third and fourth blocks. If this is not done, the directory could be overwritten when used with an unmodified BIOS, and would then appear full of garbage when used with a modified one.

"We can reserve the extra directory blocks by making the first file written to disk a dummy, full of E5 bytes. To the standard system, this will appear as a normal file; to the modified system, the E5 pattern will look like two blocks of empty directory space. To ensure that the dummy file occupies the third and fourth blocks, it must be written under a standard system and it must be the first file on the diskette. If your SYSGEN procedure copies hidden files to the new disk, the dummy file must be written before you SYSGEN the disk. The following DDT session illustrates the procedure.

```
A>ddt
DDT VERS 2.2
-f100,900,e5              fill 2K with E5h
-g0                       exit DDT
A>save 8 b:empty          save dummy file
A>stat b:empty $sys       and hide it
```

"CP/M gets its information on directory size from the Disk Parameter Block (DPB), a data area contained in the BIOS. To enlarge the directory it is necessary to alter two bytes in the DPB. The first is DRM, a count one less than the number of directory entries. The second is ALLOC0, a bit-map of the disk blocks reserved for the directory.

"The DDT session that follows shows how to find the DPB and how to make a small command file, KLUDGE.COM, to alter it. The DPB that this procedure locates is the one for the default drive. In most systems there is only one DPB for each disk format and the modification will affect all drives. If it doesn't on yours, just repeat the exercise with a different drive logged in.

```
A>ddt
DDT VERS 2.2
-a100                     assemble a program
0100 mvi c,1f             to locate the DPB
0102 call 5
0105 rst 7
0106 .
-g100                     execute it
*0105

-x                        HL has the address
C0Z1M0E1I0 A=14 B= ... H=FE14 ...
-hfe14 e                  calculate its size
FE22 FE06
-dfe14 fe22               dump it to make sure
FE14 1A 00 03 07 00 F2 00 3F 00 C0 ...
FE20 00 02 00 ...

-a100                     assemble a program
0100 mvi a,7f             to patch the DPB
0102 sta fe1b             ...change DRM to 127
0105 mvi a,f0             ...reserve 4 blocks
0107 sta fe1d             ...in ALLOC0
010A ret
010B .
-g0
A>save 1 b:kludge.com
```

"After running KLUDGE, the directory will have space for 128 entries, and should stay that way until the next cold boot, which overwrites the BIOS. Some manufacturers such as Osborne have chosen, for reasons best known to themselves, to rewrite the BIOS at warm boot also, contrary to CP/M specifications. On these systems, KLUDGE will have to be run after each boot." [*A BIOS that supports multiple formats may rebuild the DPB when a disk is selected after a warm start – DEC.*]

A Stack of Boots

Aubrey Hutchison has checked in with a warning for users of the California Computer Systems BIOS supplied with the 2422 disk controller board. "After running for 18 months," he writes, "I found a problem that was bothering me from time to time. At times when using PIP, the machine would appear to hang up; at other times it would seem to PIP forever; but most of the time PIP worked as expected.

"The problem turned out to be related to the CCS boot code and boot ROM. The stack pointer is set by the ROM to be

the top of RAM – 56Kb in my case, since at boot time the top 8Kb of RAM is banked out. When the ROM enters the cold-boot routine loaded from disk, it leaves the stack in the same place. The cold-boot routine doesn't change it. The CCS BIOS did not set the stack pointer either [*it sets it on a warm boot but not a cold one – DEC*] so until the CP/M CCP took over, the stack was not set at some other location. Since the BIOS used the stack for one or two (maybe three) levels, the stack was having a good time playing around in the BDOS. The damage must have been small since I used it in this condition for 18 months.

"My method of correcting the problem was to fix the location of the stack at the entry to the cold-boot routine. I found space to insert

```
lxi h,0100h
sphl
```

at the start of the cold-boot code without changing the original code. The same thing could be done in the BIOS [*at label BOOT*]."

What do you suppose the BDOS has in its highest two or three words that overwriting them would cause trouble, but only rarely?

The Intern's 2.2 BIOS

In the last couple of columns we talked about the fancy BIOS we built for a CP/M 2.2 system with banked storage. We have CP/M 3 running now, and in building its BIOS we had to pretty much rip up the new 2.2 BIOS and lay it down again. It seems a shame to let such a pretty thing die so soon, but we really have no more use for it. And who else would, unless they had exactly our hardware configuration? Or unless they had a masochistic desire to read approximately 95 pages of heavily commented Z80 assembly code. . . .

If you have such a masochistic desire, you can have a copy of our 2.2 BIOS for your bedside reading. Send your own 8-inch, single-density diskette and a sturdy self-addressed, postage-paid diskette mailer, and we'll duplicate the source files onto it. Address the package to "The Intern's 2.2 BIOS" c/o PCC, P.O. Box E, Menlo Park, CA 94025. Note that you should not expect to actually *use* this code. If you have any idea for doing so, be aware that it needs RMAC and LINK; it comes with minimal documentation; and it carries *absolutely no warranty or support of any kind.*

CP/M Plus – Sector Buffering

The CP/M 3 (or CP/M Plus) BDOS buffers disk sectors in storage in an attempt to speed up processing. The attempt may be successful in some cases (we haven't tried any direct-access I/O yet), but the BDOS uses its sector buffers in a way that is far from optimal for sequential I/O. This is demonstrated by some experiments we made recently.

A CP/M 3 BIOS for a banked system supplies the address of two words in storage in the Disk Parameter Header that it returns from a SELDSK call. These are the anchors of two chains of Buffer Control Blocks (BCBs). Each BCB describes a buffer that the BDOS may use to save a disk sector (a physical sector, not a 128-byte logical sector). There are separate chains for directory and data sectors. According to the *CP/M 3 System Guide,* page 8, "In a banked environment, CP/M 3 maintains a cache of deblocking buffers and directory records using a Least Recently Used (LRU) buffering scheme." And on page 46, "In general, you can enhance the performance of CP/M 3 by allocating more BCBs."

Always anxious to improve performance, we gave CP/M 3 a grand total of 71 sector buffers, each one a kilobyte, spread over our four-bank system. Seventy-one kilobytes of data space (plus another 8K of directory buffers) is not exactly a SemiDisk, but it's a lot more buffers than

(Continued on page 89)

How to make dBase II™ work magic. It's a snap with Autocode.™

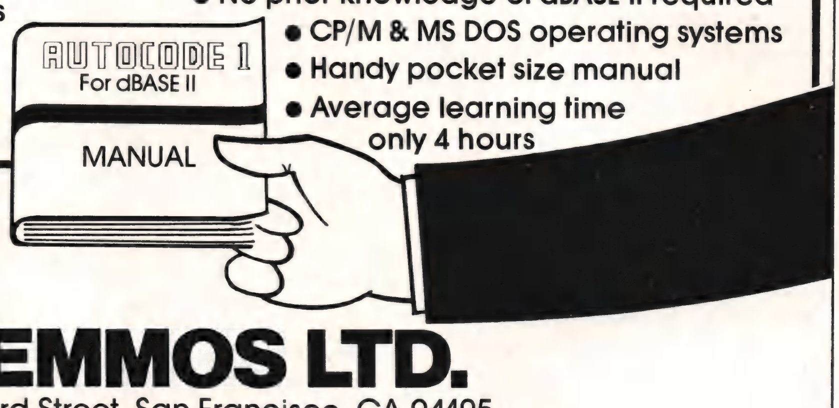
IBM/PC VERSION NOW IN STOCK

Finally, the first practical application of artificial intelligence in personal computer software.

Autocode 1 is a powerful program generator for dBASE II. No prior knowledge of programming required.

IBM is a registered trade mark of International Business Machines, Inc.

- Automatic menus & sub menus
- Automatic data entry screens
- Automatic data entry routines
- String, numeric, date & calculated fields
- Automatic multiple reports
- Automatic programs in dBASE II code with interactive screens
- No prior knowledge of dBASE II required
- CP/M & MS DOS operating systems
- Handy pocket size manual
- Average learning time only 4 hours



ONLY $200.

STEMMOS LTD.

666 Howard Street, San Francisco, CA 94105

Just send the following to address above today.

- Your diskette format & hardware
- How many Autocodes you want at $200 each*
- Your name & complete address
- A check or money order.

*In CA add 6% sales tax.

MasterCard VISA

Credit card buyers may substitute their card number and expiration date for the check. Or call us toll free and save the trip to the mail box.

ORDER TOLL FREE 800-227-1617 (Ext. 417)
IN CA CALL 800-772-3545 (Ext. 417)

U.S. Address: 666 Howard St., San Francisco CA 94105
Tel: (415) 777-3800

U.K. Address: 344 Kensington High Street, London W14
Tel: 01 602 6242

dBASE II™ Ashton Tate

Dealer inquiries invited.

Autocode 1™ Stemmos Ltd.

Circle no. 79 on reader service card.

Fast Divisibility Algorithms

It is well known that the lowest digit of an integer, if not 1, 3, 7, or 9, suffices to establish divisibility by 2, 5, or 10. Algorithms for divisibility by other small integers (such as 3, 7, 9, or 11) are not common knowledge. They require information from *all* the digits of the number, and are too complex to be part of elementary arithmetic but too trivial for the mathematical elite. Even the "obviously divisible" integers contain useful information in the next-to-last digit. E.g., numbers ending in 5 are divisible by 25 if the preceding digit is 2 or 7, while numbers ending in 0 are divisible by 25 if the preceding digit is 0 or 5. Numbers ending in 2 or 6 are divisible by 4 if the preceding digit is odd – if it is even, the number has only a single factor of 2; the opposite is true of numbers ending in 0, 4, or 8.

The algorithm for divisibility by 3 or 9 is relatively simple. It requires information from *all* the digits, but is independent of the *order* of the digits. It calculates the "reduced digit-sum" by adding all the digits; whenever this becomes 10 or more, 1 is added to the low digit and the high digit is discarded so that the sum is always a number from 1 to 9. If the final reduced sum is 9, the number is divisible by 9. If it is 3 or 6, the number has a single factor of 3. E.g., the number 123456789 has a reduced sum of 9 and so is divisible by 9. This is true of all the possible permutations of these digits, and also true for 12345678 and all its permutations. However, 1234567 has a reduced sum of 1 and so cannot be divided by 3, nor can any of its permutations. The presence of zeroes within the numerical sequence does not affect the truth of the algorithm; e.g., 45, 405, 4005 all are multiples of 9, and so of 45.

The algorithm for divisibility by 11 is somewhat more complex, since it depends on the *order* of the digits (only a fraction of the possible permutations being multiples of 11). It requires the calculation of *two* digit-sums. The "high" sum is that of the first, third, fifth, etc. digits; the "low" sum is that of the second, fourth, sixth, etc. digits. Within each of these sets, however, the order of the digits does not affect the truth of the algorithm since this requires only that the *difference* between the two sums be a multiple of 11, or 0. The difference will always be 0 if each sum is reduced by 11, though it is simpler to reduce only the difference, in one step, if it is not 0. An example is the number 12435687 with both high and low digit-sums equal to 18, and so divisible by 11. If one *adds* the two sums, the value 36 reduces to 9, so the number is divisible by 99. Which set is "high" does not matter; e.g., 21346578 is also divisible by 99, and of course also by 2. Though more complex, this algorithm can test divisibility by 3, 9, or 11 in one operation. For gigantic numbers, the digit-sums can be reduced by 99 since this is a multiple of both 9 and 11 – if a sum reaches 100 it reduces to 1, etc.

by H. T. Gordon

H. T. Gordon, College of Natural Resources, University of California, Berkeley 94720.

These algorithms are valid for number bases other than decimal, although this alters the factoring integers. In octal, successive multiples of the highest digit (7) are 16, 25, 34, 43, etc., so that numbers of any size will yield a total reduced sum of 7, while successive multiples of its base (8) + 1 (= 9) are 11, 22, 33, etc. In hexadecimal (base 16), the total reduced sum of numbers factorable by its highest digit (F, = 15 decimal) must always be F (e.g., 1E, 2D, 3C, etc.), while the "elevenish" series (11, 22, 33, etc.) has all multiples of decimal 17.

As in the decimal "base minus 1" algorithm, in which reduction by 9 reveals multiples-of-3 when the reduced sum is 3 or 6, so in hexadecimal does a reduced sum of 3, 6, 9, or 12 reveal divisibility by 3 (but not 5), while a reduced sum of 5 or A (decimal 10) reveals divisibility by 5 (but not 3). Only in decimal do multiples of 5 always have 0 or 5 as the lowest digit. The general rule (at least for bases that are even numbers) is that numbers divisible by the digit that is half the base always end in 0 or that digit.

Divisibility by 7 can now be seen to require a general "base minus 3" algorithm, that in octal would recognize divisibility by 5 and in hexadecimal divisibility by 13. I have only studied the algorithm for decimal notation. It is even more complex than divisibility by 11, requiring not only calculation of a similar high and low pair of digit-sums, but positional "weighting" of each digit. The *order* of the digits in each set is significant. Relatively few permutations within each set retain divisibility by 7, since only digits with the same weighting factor can be interchanged. Weighting involves multiplication by 1 (i.e., no change) for the first, fourth, seventh, etc. digit (reading the number from left to right), multiplication by 2 for the second, fifth, eighth, etc. digit, and multiplication by 4 for the third, sixth, ninth, etc. digit. However, digits can be reduced by 7, to a pseudo-heptal notation; i.e., 7 becomes 0, 8 becomes 1, and 9 becomes 2. Also, the product of a digit by its weighting factor can be reduced by 7; e.g., if the digit is 6 and its factor is 4, the product (24 = 3 · 7 + 3) can be replaced by 3. Since such repetitious calculation would slow the algorithm down, the operation of replacing each digit by its weighted and reduced digit can be done by the following look-up table:

Decimal Digit to be "Replaced"

Order	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	0	1	2
2	2	4	6	1	3	5	0	2	4
3	4	1	5	2	6	3	0	4	1

For example, the algorithm alters the number 421589637 to 444521660, which has 16 for both its high and low digit-sums. The original number is therefore divisible by 7 since this requires that the difference be either 0 (as in this case) or also a multiple of 7. Since this number also happens to be divisible by 9, it is divisible by 63. Another example is 10101, which alters to 10402 with a high digit-sum of 7 and a low of 0, revealing divisibility by 7 and (since the total unaltered digit-sum is 3) by 21.

All these divisibility algorithms involve no actual division. They only determine whether division by the small prime being tested is possible, and can do this relatively quickly even for gigantic numbers by reducing them to much smaller numbers that retain the same divisibility. A high proportion (56.6%) of very large random numbers ending in 1, 3, 7, or 9 will be divisible by 3, 7, or 11 and so can be proved to be non-prime. Algorithms for divisibility by higher small primes (13, 17, 19, etc.) could surely be devised, but would "sieve out" ever-decreasing fractions of non-primes at an ever-increasing cost in complexity. However, one never knows when exotic algorithms may prove useful, if only as clues to far more powerful ones.

One human-interest element is that I did a casual search for these algorithms in various math books, in the certainty that

they were known, but failed to find them. Rediscovering the 3-6-9 and the 11 algorithms – by simple inspection of the digit sequence of small multiples – was child's play. Subsequently I found these algorithms on pages 25-27 of the *VNR Concise Encyclopedia of Mathematics*, recently published (or reprinted) by Van Nostrand Reinhold. This is the work of East German mathematicians and presents a few uses in detection of errors in complex calculations. It omits the divisibility-by-7 algorithm, though I'm certain this was also discovered (probably more than once) long ago. I found it less obvious than the simpler algorithms. Its computational drudgery would, in the pre-computer era, have discouraged its use. Even the extension to other number bases has probably been done before, though like other things the elite see as trivial, it has been forgotten! The VNR Encyclopedia reference pointed out something I had ignored, that what I've called the "reduced sum" is not useless even when non-divisibility is found, since it is then the remainder that would be obtained if the division were actually done. Perhaps the divisibility-by-3 algorithm could be used to test random-number generators, since 1/3 of a fairly large sample of large numbers ought to be divisible by 3, 1/3 should have a remainder of 1, and 1/3 a remainder of 2. DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 229**

B-Tree ISAM Concepts

How do you retrieve random information quickly from a large disk file? Every programmer who has written software for business has had to deal with this problem. Fortunately, computer scientists have refined an elegant and simple method for retrieving information called a B-Tree. The objective of this article is to explain the principles and techniques involved in a B-Tree.

Why ISAM?

With current hardware technology, about the only way to quickly retrieve information out of a disk file is using an indexing method. There are many methods of indexing data. ISAM, Indexed Sequential Access Method, has become very popular for use in many business systems. It allows the information to be retrieved randomly by data value (indexed) and in sorted order (sequential) using just one index file. This dual ability is what distinguishes ISAM from other methods, such as Hashing. While there are many ways to implement an ISAM index, the B-Tree is generally accepted as being the current "state of the art."

General ISAM Principles

An ISAM file is a file composed of individual pieces of information called "keys." A key is an ASCII string representing some value in a data record. The index is arranged in such a way that keys can be retrieved randomly and sequentially, along with their associated data record numbers. There are a number of different schemes designed to serve this purpose, a B-Tree being only one. To better appreciate the B-Tree, we will look at an older method, called the binary tree.

A tree structure is called such because if all the search paths are drawn out, they resemble an inverted tree as shown in Figure 1 (at right). The search starts at the root and progresses down the tree until it possibly reaches the bottom (called leaves, or leaf level). Note also in Figure 1 that from any point in the tree, there are two paths to the next lower level, hence the term "binary tree." In a simple binary tree, each key is stored in an individual record, or node. Each node also has two pointers to other nodes. These pointers are what make up the search paths through a tree.

A search through a binary tree is, on a basic level, a simple procedure. To find the key "G," it is first compared to the key in the root node. If "G" matches, then the search is successful. Otherwise, a decision must be made, that is, where to go next. If "G" is smaller than the root node, then the path to the left is taken. If "G" is larger than the root, then the path to the right is taken. The path is followed and the new node is read. "G" is then compared to the key in this node as before, and the appropriate action is taken. The search continues until either "G" is found, or a leaf node is unsuccessfully evaluated.

As long as the nodes are kept in memory, the binary search is an efficient method. Once the nodes are stored on the disk, however, the performance quickly degrades because of the large number of disk accesses required.

Other problems can also arise when keys are inserted. This can have the effect of making the tree unbalanced – that is, some search paths being longer than others. This requires that the search and insertion algorithms be aware of this possibility, and subsequently these procedures become much more complicated.

The Basic B-Tree

The problems with the binary tree in large disk-based filing systems gave an incentive for researchers to look for something better. In the late 1960s a number of people independently designed such a method called a B-Tree. Apparently the reason for the name, B-Tree, seems to be the fact that R. Bayer, then at Boeing Scientific Research Labs, was one of the pioneers of the method ("Bayer-Tree").[1]

Figure 2 (page 19) shows an example of the B-Tree format. The most obvious difference between a binary tree and a B-Tree is that from any one node there can be more than two paths to the next node. This has the effect of allowing many more keys in each node of a B-Tree than with a binary tree. For instance, a binary tree with ten-byte keys, holding one million keys, could take as many as twenty node searches to find any one key. A B-Tree with the same conditions (assuming ten keys per node) would at the most take five node searches. In the case where node searches correspond to disk accesses, the search time difference is obviously dramatic.

Searching a B-Tree

To find a key in a B-Tree, the first step is to look at the root node. In the

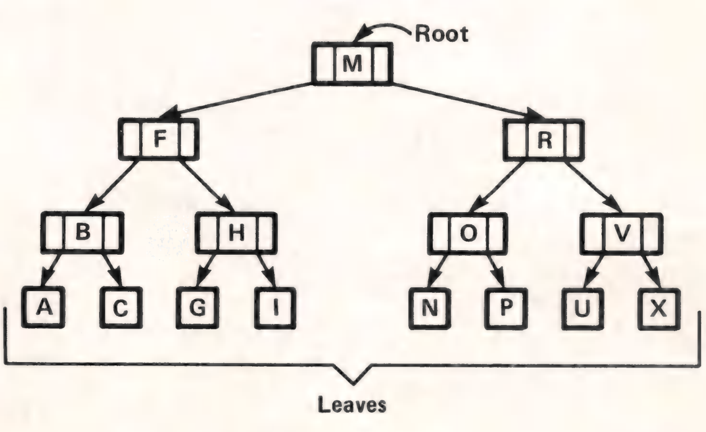


Figure 1. A Binary Tree

by Chris Deppe and Alan Bartholomew

Alan Bartholomew, Business Computing, 2210 Wilshire Blvd., Suite 289, Santa Monica, California 90403.

Chris Deppe, 6333 Canoga Avenue #191, Woodland Hills, California 91367.

example shown by Figure 2, there are a maximum of three keys in each node. To decide what to do, a process called scanning is used. This involves sequentially looking at each key in the node (note they are stored in ASCII sequence) and stopping when either a matching key is found (in this case, the search is successful, and it ends) or a higher key is found. If no match is found, then a decision must be made.

Take, for example, a search for the key "L." By scanning the root node in Figure 2, we find that there is no matching key. We therefore must follow a pointer to the next lower level. The pointer to follow is the one which sits where the key "L" would be if it existed in the node. In this case we would follow the middle pointer to the next lower node. This node is then read and the same procedure is followed. This continues until either the key is found, or a leaf node is unsuccessfully scanned.

Inserting into a B-Tree

Note that since we must determine where the key would sit if it existed in the node, we now have the needed information to insert the key. Insertion into a B-Tree uses the search already described. Then the key is simply inserted into the node where it should be. Note that keys are always inserted into leaf nodes (an insertion depends not on finding the key, but on finding a place for it). Since an unsuccessful search always ends at the leaf level, then all keys will be inserted there.

If the node into which the key is to be inserted is full, then a split occurs. The original node and the new key are divided up into two new nodes. Since there is now a new node, a pointer to the new node must be inserted into the level above. Usually the middle key of the two new nodes is brought up to the next level to be used as a separator. If the node above is also full, then it too might be split. This can continue up to the root. If a split of the root node occurs, a new root node is created so that the tree becomes one level higher.

A B-Tree by nature is always a balanced tree. Since all node expansions are done on the same level, the tree never gets unbalanced. An insertion will never increase the search path to one leaf node and not the others.

Deletion of a key is simply finding the key and taking it out of the node. If a key doesn't reside in a leaf, then a new key must take its place to provide the same paths as the deleted key. This new key is found by getting the next key in sequence from the deleted key.

As stated before, the search time in a B-Tree is much more efficient than in a similar binary tree when stored on disk. Sequential processing in a B-Tree is not so easy, however. Because the keys are distributed between all levels of the tree, keys must be retrieved by following the links up as well as down the tree, working from the left to right. This becomes awkward and slow. Try it by hand using Figure 2 and you will quickly see why.

If the sequential processing of a B-Tree could be improved while retaining the random search efficiency and balanced nature, we would have a much better method.

The B+Tree

This brings us to one of the most important variants of the B-Tree called the B+Tree. Figure 3 (page 21) shows an example of the B+Tree format. In a B+Tree, all the keys are stored in leaf nodes. The upper levels simply provide pointers to the next lower level, and so on until the leaf level is reached. In addition, all the leaves are linked together. What we have, in effect, is a tree which provides a B-Tree type of path to the proper position in a sequential list of the keys.

Since all the keys reside on the leaf level, sequential processing is now very easy as each key is linked together sequentially in the leaf nodes. This gives the B+Tree the desired additional characteristics of an ISAM while retaining the simplicity of a normal B-Tree.

The B+Tree search uses the upper levels as a roadmap to the next level, and it is only until a leaf is reached that the key is actually looked for. Therefore, all searches use the same number of node reads. Although this guarantees that every search will be a "worst case" of a normal B-Tree, we have seen that this worst case is very good under most circumstances. In addition, since all searches take approximately the same amount of time, a high degree of consistency is achieved, which can have its benefits in a real-world situation.

One nice benefit of the B+Tree structure is that since the upper levels are just pointers, a delete doesn't have to worry about revising pointers (deletes only operate on leaves which have no pointers).

Key searching and insertion methods are basically the same as in a normal B-Tree. The specific B+Tree we implemented uses a scheme which uses the high key from the left node to provide a separator (located in the index node above) between the left node and the right node. For example, in Figure 3, a search for the key "H" would produce the following. First, "H" is compared to the first key in the root node. Since "H" is less than "I" the search proceeds by reading the next

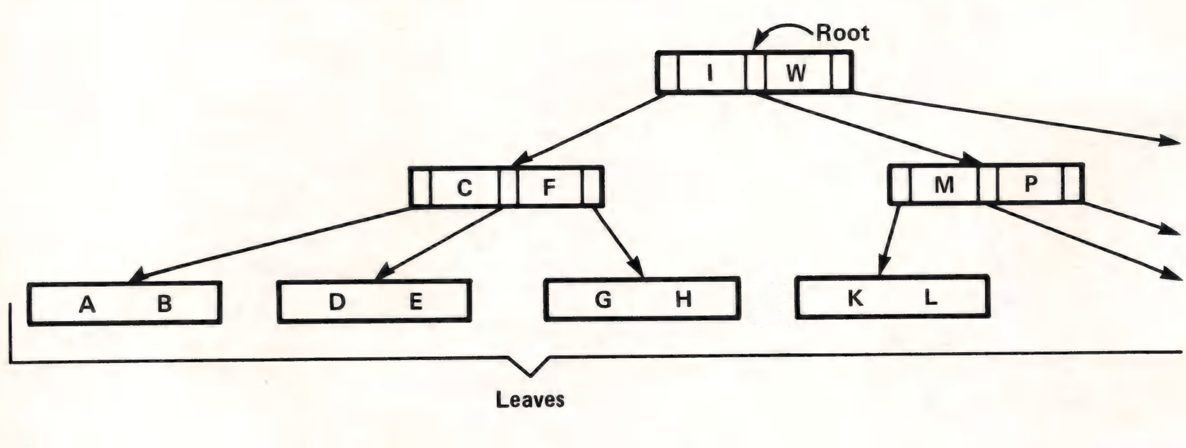


Figure 2. A B-Tree

lower node using the left-most pointer. Then "H" is compared with the first key in the lower node. Since "H" is not smaller than or equal to "C," we must look at the next key. Again, since "H" is not smaller than or equal to "F," we move on to the next key. The next key is "I." "H" is smaller than "I." Since "I" represents the largest value stored in the node its left pointer points to, we follow that pointer.

Splits also use this same high-key logic. At split time, the high key is taken from the left node and it is then inserted (with a pointer to the right node of the split) in the index node above (see Figures 4a and 4b on page 21).

One thing should be mentioned about the B+Tree. Since all the keys are in leaves, the nodes above represent an additional overhead not found in the basic B-Tree, which uses only key values in its tree. With the advent of inexpensive mass storage, though, a slight bit more storage space taken up is well worth the many benefits a B+Tree has to offer.

B-Trees on Micros

Many implementations of the B-Tree ISAM have been created. For example, IBM uses a variant of the B-Tree for their mainframe computers called "VSAM." More recently, the B-Tree ISAM techniques have been applied to microcomputer applications. For example, dBase II uses a B-Tree method for indexing. Also, a number of companies have developed B-Tree subroutine products for microcomputers. These subroutines can be added to an application being developed. The application programmer uses call statements to the indexing functions built into the B-Tree subroutine without having to be concerned with all the details of how it works.

B-Tree implementations have been produced to support a wide variety of languages. The authors have developed a version written in Forth. Other languages supported by various B-Tree packages include BASIC, Pascal, C, COBOL, Fortran, and others.

For the programmer interested in creating a B-Tree "from scratch," the references for this article will be helpful. These references provide more details about B-Tree techniques and the many subtle variations possible in designing a B-Tree ISAM.

The authors have implemented B-Tree ISAM technology in a Forth utility called INDEX+. Written for Laboratory Microsystems Forth, the package includes fully documented source code and a manual which includes tutortials on B+Tree technology and use of the utility. For more information, contact Laboratory Microsystems.

References

[1] Knuth, Donald E. *The Art of Computer Programming, Volume 3, Sorting and Searching,* Addison-Wesley Publishing Company, 1973.

[2] Comer, Douglas, "The Ubiquitous B-Tree," *Computing Surveys,* Volume 1, No. 2, June 1979.

DDJ

(Figures 3 and 4 at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 231**

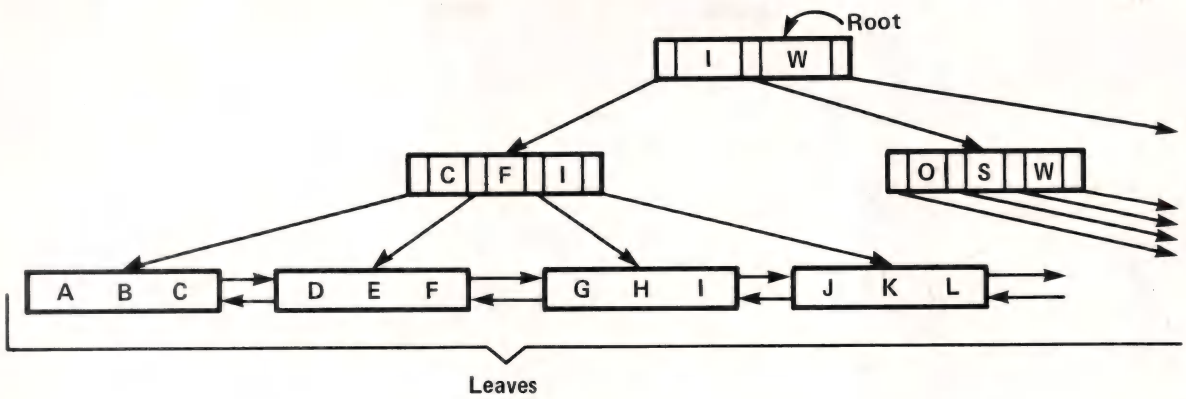


Figure 3. A B+Tree

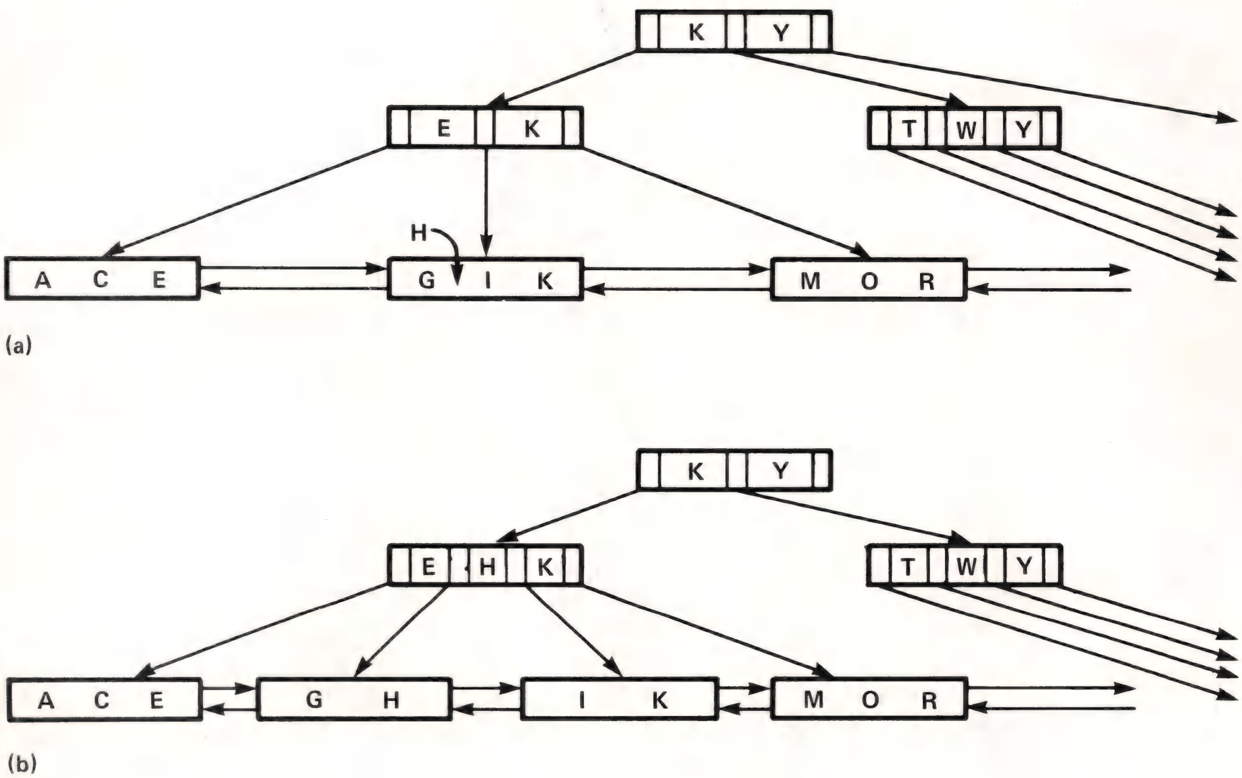


Figure 4.

Figure 4(a): In order to insert the key "H," the mode to receive it must be split. Figure 4(b): The tree after the split. Note that the new pointer in the upper node is the high key from the left node of the two new nodes.

CP/M BDOS and BIOS Calls for C

If you should want to write CP/M utilities in C, you will probably need facilities to access the BDOS (Basic Disk Operating System) and/or the BIOS (Basic Input/Output System). Many C compilers for CP/M do not include such functions, and several of those which exist have severe limitations, in that they may not return proper values in all cases.

The two functions bdos() and bios() described here will enable you to incorporate direct BDOS and/or BIOS calls in programs written in C. Please note that programs that call bdos() or bios() will not be portable beyond CP/M and the 8080 and Z80 series CPUs.

by Terje Bolstad

Terje Bolstad, Elektrokonsult AS, Konnerudgt. 3, N-3000 Drammen, Norway.

bdos() and bios() were originally written for the C/80 compiler from The Software Toolworks, but they will also compile/assemble/work with other compilers which push arguments on the stack in a non-reversed order before calling a function. Such C compilers include Small-C and most of its derivatives. bdos() and bios() have been tested to work properly with both The Software Toolworks C/80 and The Code Works CW/C compiler.

bdos ( )

The bdos() function sets machine register C to the function number specified in funct, and the register pair DE to the value given in arg, and initiates a call to BDOS. The function number may be specified numerically or symbolically. No checking of the arguments is done.

If funct is either RETVN (12), RETLV (24), GETAA (27), GETROV (29) or GETDPA (31), bdos() returns the value remaining in register pair HL on return from BDOS. For all other BDOS functions, bdos() returns the value in A, with sign extension. In this way, BDOS error (255 or FF(hex)) is returned as –1.

bios ( )

The bios() function sets machine register pair BC to the value given in arg1, and the register pair DE to the value given in arg2, and initiates the appropriate BIOS call by transferring control to the BIOS jump vector entry point specified in funct. This entry point may be specified numerically or symbolically. No checking of the arguments is done.

If funct is either SELDSK (9) or SECTRAN (16), bios() returns the value remaining in the HL register after execution of the BIOS call; otherwise it returns the value remaining in register A on return from BIOS, with sign extension.

Note that you must specify all three arguments in the bios() function. If you do not need the last one, you must set it to 0 or any other value.

Installation

You may type the source code in Listing 1 (page 24) into a file called CPMCALL.C. To use bdos() or bios() in a C source file, you must #include "CPMCALL.C" in that file.

A simple example on how to use bdos() and bios() is shown in Listing 2 (page 27). In this example, bdos() is used to get the current disk and bios() is used to print it on the console. You may use this example to test that you have installed bdos() and bios() correctly on your system. When run, the program should write the uppercase character corresponding to the current (logged) disk drive on the console. Log onto different drives and check that the program writes out the correct letter.

Normally it is not advisable to do input/output via bios() or bdos() (as done with bios() in the example), if this can be done via other conventional C functions, such as putchar().

In order to be compatible with code for other compilers which do offer the same functions (e.g., SuperSoft's C compiler), you should refer symbolically to the bdos() and bios() function numbers (as shown in the listings) and never use numbers directly. The reason for this is that other compilers may not use the same numerical values for bios() function numbers as used in this version of bios().

You may want to delete (or "comment out") all the #define macros which will not be used, so that they will not occupy unnecessary space in the compiler's macro substitution table.

After compilation, bdos() and bios() will assemble under CP/M's ASM, The Software Toolworks' AS, and most other 8080 assemblers. Both functions may be assembled for nonstandard versions of CP/M by changing the CPBASE equate from 0 to the required value.

If you should want to delete the bdos() function and make a separate function of bios() only, you need to include the CPBASE equate in front of the bios() assembly code. To use bdos() and bios(), refer to the CP/M Interface Guide and the CP/M Alteration Guide.

A final warning: You should avoid direct CP/M calls whenever possible. Even though they enable you to do wonderful things in CP/M, they will limit the portability of your programs.

The code may be copied, used, and distributed by anyone, commercially or otherwise, provided the contribution notice is included.

DDJ

(Listings begin on page 24)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 233**

Programmers...Now you can "C" into your programs!

C Compiler

Runs on the IBM® Personal Computer or MSDOS® compatible computers. Provides features and flexibility needed for both systems and application programming. Makes it possible to test programs at the C source level. The software program features full documentation, generates fast code, and has a standard C function library.

- All C language elements for operations on characters, signed and unsigned integers, structures, and unions
- Outputs IBM PC assembler source code
- PL/M compatibility mode
- Embedded shell in compiled output supports argc, argv, and I/O redirection
- Complete preprocessor
- Allows more than 64K of code; extended pointers allow access to more than 64K of data

c-window™ Source Level Debugger

Designed for sophisticated programmers who want more capability than inserting "printf" statements or using assembler level debugging . . . new programmers who want to better understand how C operates . . . trainers who teach the C language. It helps you find the precise statement in a program where an error occurs and observe when data values are changed in a program.

- True source level single step
- Auto and external variable display and modification in a variety of formats
- Full C expression evaluation and recursion
- Varied breakpoints
  - —at function name and line number
  - —sticky and non-sticky
  - —at function whenever an expression evaluates as TRUE
- Multiple variable display as program is tested

c-systems

The following is an excerpt from a typical c-window debugging session. The ">>" is the c-window prompt, and the underscored text is the operator input. The text on the right is a brief description of the operations taking place.

```
entry at main line 15              ▪ c-window entry prompt showing function
                                     name and line number
>>bs addline, 58                   ▪ A breakpoint is set in function addline,
                                     line 58.
>>g                                ▪ The "go" command starts execution.
break at addline line 58           ▪ The breakpoint is reached.
>>d ptr                            ▪ The value of variable "ptr" is displayed.
bae
>>d ptr = root                     ▪ The variable "ptr" is set to the value of the
f02                                  variable "root".
>>d ++ptr                          ▪ The variable "ptr" is incremented to the next
f14                                  entry in the table.
>>ds ptr–>symname                  ▪ Display the string at the member "symname"
xyzzy                                in the structure pointed to by "ptr".
>>cs 1 d i                         ▪ Command set. The variable "i" will be
                                     displayed for each break in execution.
>>s                                ▪ Single step.
step at addline line 59            ▪ c-window displays the next line to be
                                     executed.
<1> d i                            ▪ c-window executes the automatic command,
11                                   showing the value of "i" to be 11 hex.
>>d i==j                           ▪ Test if "j" has the same value as "i".
1                                  ▪ They are equal.
>>bx addline,2,i<j&& k==0.         ▪ Set an expression break. Once execution
                                     begins, c-window will break execution on
                                     the second occurence of the expression
                                     "i<j&&k==0" evaluating non-zero in the
                                     function "addline".
```

For the finest time-saving software contact:

c-systems We make programming easier

P.O. Box 3253 • Fullerton, CA 92634 • (714) 637-5362

Name ______________________________

Address ______________________________

____

I would like to order:

- ☐ C Compiler. $195.00
- ☐ c-window (requires C Compiler) $195.00
- ☐ Demo package (includes diskette, manual) price applicable toward purchase of above . . $45.00

Require 128K and two disk drives.
c-window is a trademark of C-Systems. IBM is a trademark of IBM Corp. and MSDOS is a trademark of MicroSoft.
Prices subject to change without notice.

BDOS and BIOS Calls for C

Listing One (Text begins on page 22)

```
/************************************************
*                                               *
*  CP/M BDOS- AND BIOS- CALLS FOR C/80          *
*                                               *
*  Contributed by T. Bolstad, ELEKTROKONSULT AS *
*  Konnerudgaten. 3, N-3000 Drammen, NORWAY.    *
*                                               *
*  Date:  January 17, 1983.                     *
*                                               *
************************************************/




/*  DEFINITION OF BDOS FUNCTIONS                  */

#define RESET     0    /*  SYSTEM RESET             */
#define CONSIN    1    /*  CONSOLE INPUT            */
#define CONSOUT   2    /*  CONSOLE OUTPUT           */
#define READIN    3    /*  READER INPUT             */
#define PUNOUT    4    /*  PUNCH OUTPUT             */
#define LISTOUT   5    /*  LIST OUTPUT              */
#define DIRCON    6    /*  DIRECT CONSOLE I/O       */
#define GETIOB    7    /*  GET I/O BYTE             */
#define SETIOB    8    /*  SET I/O BYTE             */
#define PRNTST    9    /*  PRINT STRING             */
#define READCB   10    /*  READ CONSOLE BUFFER      */
#define GETCST   11    /*  GET CONSOLE STATUS       */
#define RETVN    12    /*  RETURN VERSION NUMBER    */
#define RESDSK   13    /*  RESET DISK SYSTEM        */
#define SELDISK  14    /*  SELECT DISK              */
#define OPENF    15    /*  OPEN FILE                */
#define CLOSEF   16    /*  CLOSE FILE               */
#define SRCHFF   17    /*  SEARCH FOR FIRST         */
#define SRCHFN   18    /*  SEARCH FOR NEXT          */
#define DELF     19    /*  DELETE FILE              */
#define RDSEQ    20    /*  READ SEQUENTIAL          */
#define WRSEQ    21    /*  WRITE SEQUENTIAL         */
#define MAKEF    22    /*  MAKE FILE                */
#define RENF     23    /*  RENAME FILE              */
#define RETLV    24    /*  RETURN LOGIN VECTOR      */
#define RETCD    25    /*  RETURN CURRENT DISK      */
#define STDMA    26    /*  SET DMA ADDRESS          */
#define GETAA    27    /*  GET ALLOCATION ADDRESS   */
#define WPDSK    28    /*  WRITE PROTECT DISK       */
#define GETROV   29    /*  GET READ/ONLY VECTOR     */
#define SETFAT   30    /*  SET FILE ATTRIBUTES      */
```

```
#define GETDPA  31   /*  GET DISK PARAMETERS ADDRESS  */
#define SGUC    32   /*  SET/GET USER CODE          */
#define RDRAN   33   /*  READ RANDOM                */
#define WRRAN   34   /*  WRITE RANDOM               */
#define COMFS   35   /*  COMPUTE FILE SIZE          */
#define SETRRC  36   /*  SET RANDOM RECORD          */
#define RESDRV  37   /*  RESET DRIVE                */
#define WRRZF   38   /*  WRITE RANDOM WITH ZERO FILL  */

bdos(funct,arg)      /* corresponds to bdos((BC),(DE)) */
int     funct,arg;

/*  CALL EXAMPLE:    bdos(RETVN,0)
    BOTH ARGUMENTS MUST BE SPECIFIED !
    Values are returned IN HL.  BDOS errors
    are returned as -1.                         */

{
#asm
CPBASE  EQU     0          ;NORMAL 0-ORG'ED CP/M
CPNTRY  EQU     CPBASE+5 ;BDOS ENTRY

        POP     H          ;GET RETURN ADDRESS
        POP     D          ;GET ARG (INFORMATION ADDRESS)
        POP     B          ;GET FUNCTION NO.
        PUSH    B          ;RESTORE STACK
        PUSH    D
        PUSH    H

        PUSH    B          ;SAVE FUNCTION NO. ON STACK
        CALL    CPNTRY     ;BDOS CALL
        XCHG               ;SAVE HL IN DE
        MOV     L,A        ;SAVE A IN L
                           ;SIGN EXTENSION TO H:
        RLC                ; GET SIGN BIT INTO CY
        SBB     A          ; IF CY=0, RESULT AFTER SBB IS ZERO
                           ; IF CY=1, RESULT AFTER SBB IS -1 (IE ALL ONES)
        MOV     H,A        ; NOW A IS MOVED TO HL WITH SIGN EXTENSION
        POP     B          ;GET FUNCTION NO IN BC
        MOV     A,C        ;GET FUNCTION NO IN A

        CPI     12         ;WAS IT 'RETURN VERSION NUMBER' ?
        JZ      RETHL1
        CPI     24         ;RETURN LOGIN VECTOR ?
        JZ      RETHL1
        CPI     27         ;GET ALLOCATION ADDRESS ?
        JZ      RETHL1
        CPI     29         ;GET READ/ONLY VECTOR ?
        JZ      RETHL1
        CPI     31         ;GET DISK PARAMETER ADDRESS?
        JZ      RETHL1
        JMP     BDOSRET
```

(Continued on next page)

BDOS and BIOS Calls for C

Listing One (Listing continued, text begins on page 22)

```
RETHL1: XCHG
BDOSRET: RET                    ;WITH RETURNED VALUE IN HL

#endasm
}

/*      DEFINITION OF BIOS FUNCTIONS      */

#define BOOT      0    /*  COLD-BOOT                */
#define WBOOT     1    /*  WARM-BOOT                */
#define CONST     2    /*  CONSOLE STATUS           */
#define CONIN     3    /*  CONSOLE INPUT            */
#define CONOUT    4    /*  CONSOLE OUTPUT           */
#define LIST      5    /*  LIST DEVICE              */
#define PUNCH     6    /*  PUNCH                    */
#define READER    7    /*  READER                   */
#define HOME      8    /*  HOME DISK DRIVE HEAD     */
#define SELDSK    9    /*  SELECT DISK DRIVE        */
#define SETTRK   10    /*  SET TRACK                */
#define SETSEC   11    /*  SET SECTOR               */
#define SETDMA   12    /*  SET DMA ADDRESS          */
#define READ     13    /*  READ ONE SECTOR          */
#define WRITE    14    /*  WRITE ONE SECTOR         */
#define LISTST   15    /*  LIST STATUS              */
#define SECTRAN  16    /*  SECTOR TRANSLATION       */

bios(funct,arg1,arg2) /* corresponds to bios(function,(BC),(DE)) */
int funct,arg1,arg2;

/*  CALL EXAMPLE:     bios(SETTRK,5,0)
ALL 3 ARGUMENTS MUST BE SPECIFIED, even though
the last one is only used by SELDSK and SECTRAN. */

{
#asm

        POP     D       ;RETURN ADDRESS
        POP     H       ;ARGUMENT 2
        SHLD    ARG2S   ;SAVE IT
        POP     B       ;ARGUMENT 1
        XCHG            ;GET RETURN ADDRESS INTO HL

        POP     D       ;FUNCTION NO.

        PUSH    D       ;RESTORE SP
        PUSH    B
        PUSH    B
        PUSH    H       ;RESTORE RETURN ADDRESS

        PUSH    D       ;SAVE FUNCTION NO. ON STACK

        LXI     H,0     ;CALCULATE OFFSET ADDRESS FROM FUNCTION:
        DAD     D       ; GET FUNCTION NO. (OFFSET) IN HL
```

```
        DAD     H         ; 2*OFFSET
        DAD     D         ; 3*OFFSET
        XCHG              ; SAVE OFFSET ADDRESS IN DE

        LHLD    CPBASE+1 ;GET POINTER TO BIOS WBOOT ENTRY
        DCX     H         ;DECREMENT TO
        DCX     H         ;  POINT TO
        DCX     H         ;     START OF BIOS ENTRY JUMP TABLE

        DAD     D         ;ADD OFFSET (RESULT IN HL)
        XCHG              ;GET RESULT IN DE
        LXI     H,RET1
        PUSH    H         ;SAVE RETURN ADDRESS ON STACK

        LHLD    ARG2S     ;GET ARGUMENT 2
        XCHG              ;GET ARGUMENT 2 INTO DE
                          ; AND BIOS FUNCTION ENTRY ADDRESS INTO HL

        PCHL              ;GO TO BIOS

RET1:   XCHG              ;SAVE HL IN DE
        MOV     L,A

        RLC               ;GET SIGN BIT INTO CY
        SBB     A         ;IF CY=0, RESULT AFTER SUBB IS ZERO
                          ;IF CY=1, RESULT AFTER SUBB IS -1 (IE ALL ONES)
        MOV     H,A
        POP     B         ;GET BIOS FUNCTION NO. IN BC
        MOV     A,C
        CPI     9         ;SELECT DISK FUNCTION ?
        JZ      RETHL2
        CPI     16        ;SECTOR TRANSLATION FUNCTION ?
        JZ      RETHL2

        JMP     RETBIOS
RETHL2: XCHG              ;RETURN VALUE IN HL
RETBIOS: RET
ARG2S:  DS      2

#endasm
}
```

End Listing One

Listing Two

```
#include "cpmcall.c"
main()
{
        bios(CONOUT,bdos(RETCD,0)+'A',0);     /* print current disk */
}
```

End Listing Two

GREAT DRIVES. GREAT PRICES.

Floppy Disk Services offers both.

We supply the industry with high quality disk drives and peripherals. And our prices are some of the best you'll see. Floppy Disk Services is a contracted dealer for Siemens, Tandon and Shugart. Starting with their drives, we design complete packages to give you the system capacity you're looking for and the dependability you need.

We offer add-on drives for IBM, Radio Shack, Heath, Apple and most other microcomputers, all at a significant savings to you. Check the examples below and you'll see what we mean.

Apple II Add on drives	$300.00
Apple 8 inch controller	315.00
Apple dual 8 inch system w/controller	1165.00
FDD-100-5b 'flippy' exact HEATH add on	215.00
FDD-200-5 double sided 40 track drive	250.00
SA 455 half hgt DS/DD 48TPI ... $295.00	2 for $245.00 each
SA 465 half hgt DS/DD 96TPI ... $350.00	2 for $295.00 each
FDD-221-5 5ms step 80 track DD/DS	330.00
FDD-100-8d 8" single side DD drive .... $240.	2 / $225. each
FDD-200-8p Double sided 8" drive .... $340.	2 / $325. each
TM-100-2 (IBM)	265.00
SA-860 DS/DD Half Hgt 8" $495.00	2 for $445.00 each
System packages available for all drives	
Dual 8 inch system with EVERYTHING	750.00*
Dual double sided 8 inch system	950.00*
Single 5¼ Heath or MOD I Add on w/case	265.00*
Dual 5¼ Heath or MOD I	505.00*
10mb Hard Disk for any computer	2300.00*
Magnolia controller, allows any combo 8 and 5¼ inch drives to be added to your H88 or H89	525.00

* 8 inch systems are fully assembled and tested, however the drives are shipped separately from the case to comply with UPS weight restrictions. All 5¼ inch systems come assembled and tested. Prices and specs subject to change without notice.

There's much more. If you don't see what you want, give us a call between 9 am and 5 pm (ET). Chances are we'll have what you need at your price.

Featuring the Shugart Thinline Series.

Floppy Disk Services carries the new Shugart Thinline Dual 8 Inch Drive (Model SA-860). It's a double-density drive, with dual head for a storage capacity of 1.25 megabytes per drive. The system includes our custom cabinets, comes fully assembled and tested, and uses the Power One CP-206 supply, the standard of the industry. Special options available: external drive select using DIP switches, and cable connector on the rear of the cabinet for ease in connecting to your system. Floppy Disk Services designs and builds these systems from the ground up to maximize efficiency and minimize space requirements. Available with Tandon Drives also.

Custom Enclosures. At Floppy Disk Services, we also sell disk drive enclosures, designed by our own experts to be functional and attractive. And our quantity pricing is so competitive, we invite dealers and group purchasers to call.

Thinline/Half Height Specials. Floppy Disk Services has just contracted with Shugart, and we're carrying their half-height 5¼ & 8" systems. If you would like some unbelievable prices on SA-455, SA-465, SA-860's and others, call today! We are legitimate contracted dealers—no middle-man.

Due to production deadlines, prices in this ad are 2 months old, so we encourage you to call us for current prices and new product info.

PAYMENT POLICY — We accept MasterCard, VISA, personal checks & MO. We reserve the right to wait 10 working days for personal checks to clear your bank before we ship. All shipping standard UPS rates plus shipping & handling. NJ residents must add 6% tax.

Free Catalogue. If you'd like to receive our Catalogue of Disk Drives and Peripherals, just call or write — we'll mail your copy immediately. And if you want to talk with an expert about getting more out of your system, we'll be happy to help.

Toll Free Order Line **(800) 223-0306**
Tech Help or Info. **(609) 799-4440**

FLOPPY DISK SERVICES INC. ™

741 Alexander Road Princeton, NJ 08540

Circle **no. 31** on reader service card.

C compilers and Cross compilers

TELECON'S C COMPILERS OFFER YOU

- FULL C
- UNIX* Ver. 7 COMPATABILITY
- NO ROYALTIES ON GENERATED CODE
- GENERATED CODE IS REENTRANT
- C AND ASSEMBLY SOURCE MAY BE INTERMIXED
- UPGRADES & SUPPORT FOR 1 YEAR

IN THESE CONFIGURATIONS:

HOST	6809 TARGET	PDP-11*/LSI-11* TARGET	8080/(Z80) TARGET	8088/8086 TARGET
FLEX*/UNIFLEX* **OS-9***	$200.00 WITHOUT FLOAT $350.00 WITH FLOAT	500.00	500.00	500.00
RT-11*/RSX-11* PDP-11*	500.00	200.00 WITHOUT FLOAT 350.00 WITH FLOAT	500.00	500.00
CP/M* 8080/(Z80)	500.00	500.00	200.00 WITHOUT FLOAT 350.00 WITH FLOAT	500.00
PCDOS*/CP/M86* 8088/8086	500.00	500.00	500.00	200.00 WITHOUT FLOAT 350.00 WITH FLOAT

Others Pending

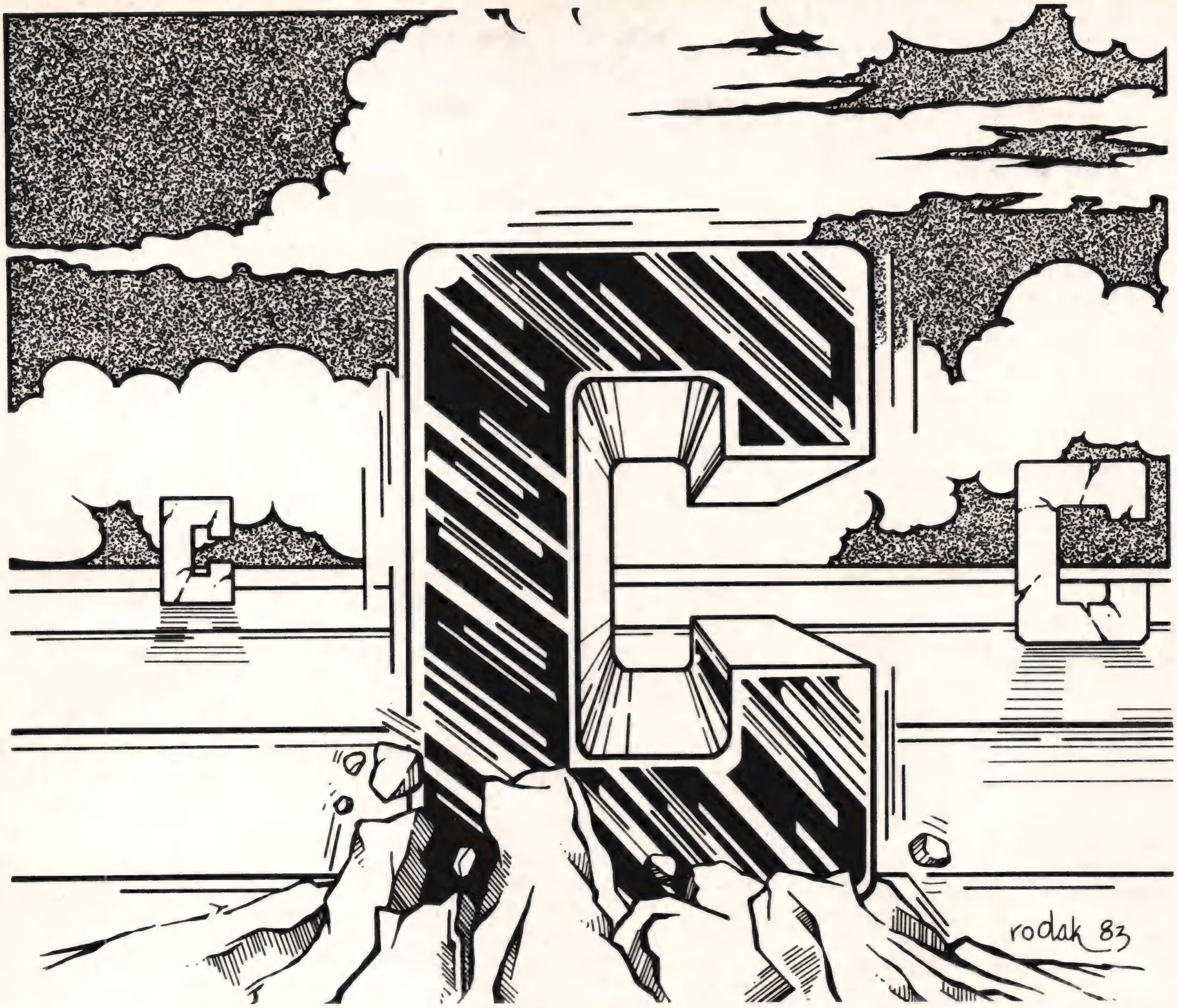
C SOURCE AVAILABLE FOR $2,500[00]

SO . . . IF YOU'RE READY TO MOVE UP TO C . . .

CALL

408-275-1659

TELECON SYSTEMS
1155 Meridian Avenue, Suite 218
San Jose, California 95125


rodak 83

Printing Graphics Using the IBM PC

The combination of the Color/Graphics Adapter and a printer capable of "dot graphics" provides a powerful utility function. The figure on page 36 shows an example of the image produced. PC-DOS at levels v.1.0 and v.1.1 does not provide direct support for this function. As this article is being written, v.2.0 was announced. I've heard that the new release corrects this omission, but my local suppliers can't supply v.2.0 as yet, so I'll ignore this new capability. All references to PC-DOS in this article should be assumed to refer to v.1.0 or v.1.1.

I've attempted to keep this at a "general" level, so that those of you who have "different" hardware may use the information to write your own utility. Specifically, however, the article (and included listing) is written for the standard IBM Color/Graphics Adapter and the original IBM Printer (EPSON MX-80) *with the GRAFTRAX PLUS ROM installed!* As I'm sure many of you are aware, the MX-80 produces quite variable results depending upon which control ROM is installed. The Graphics printer, provided on later models, should also work "as is," but minor adjustments may be required.

To introduce the "solution," it is first necessary to define the problem. In its simplest form, the problem is that of mapping. The color/graphics regeneration buffer contains a bit-encoded form of the displayed image. Dot matrix printers also use a bit-encoded form of the image to be printed. As you might expect, the two encoded forms are not even similar. The requirement for this utility, then, is simply that of mapping one bit-encoded form to another. Of course, before mapping can be performed, it is first necessary to understand the "rules" for encoding in both the input and output forms.

by Dan Daetwyler

Dan Daetwyler, Route 5, Box 518A, Springdale, Arkansas 72764.

Display Regeneration Buffer Format

For detailed specification of the input form, you are referred to the *IBM Technical Reference Manual* for the PC. This article considers only the medium-resolution graphic mode, and includes the basic information that describes the bit stream used in this mode. In this mode the addressable unit is the "pel," and the screen image is composed of 64,000 pels. The pels are "compressed" so that each byte of the buffer contains four pels, hence the regeneration buffer size is 16,000 bytes. Due to the compression, each pel must be two bits in size, giving four possible bit patterns (0 through 3). The zero pattern indicates the "background" color, while patterns 1 through 3 indicate which of the three possible "foreground" colors has been selected. Palette selection and background color selection are not germane to this article, so I'll ignore them.

So far, this encode seems simple. You need only visualize a string of bytes which are associated with the screen positions. The first byte of the buffer is the upper left corner of the screen. The four pels corresponding to the first four dots (left to right) at the upper left corner of the screen are in this byte. The second byte contains the next four dots, etc., until 320 dots have been defined (80 bytes). Unfortunately, this simple and straightforward representation now breaks down. The second row of dots on the screen is not the next 80 bytes of the buffer, but rather is the 80 bytes at an offset of 8000 in the buffer. The regeneration buffer is organized into two blocks or banks of 8000 bytes each. I won't bore you with the rationale behind this organization, since it's purely related to hardware design and addressing. From our viewpoint it's sufficient to know that even-numbered rows are stored in the first bank, while odd-numbered rows are in the second bank.

To summarize, medium-resolution graphic images appear in the regeneration buffer as they appear on the screen, left to right, top to bottom, but the pels are compressed four to a byte, and alternate rows are stored in alternate banks.

Print Line Buffer Format

Dot matrix printing, on the other hand, considers a print "line" to contain eight rows of dots, left to right, top to bottom. The most significant bit of each byte corresponds to the top row of dots in the print line. The printer used for the program provided offers two forms of dot matrix print: normal and high density. High density gives maximum resolution, so that was chosen. When operating in dot matrix, high-density mode, the printer will record 960 columns in the maximum print line (a column is a "dot"). Although one could map the graphic screen on a pel-to-dot basis, the image would be small, and no simulation of color would be possible. Since the maximum screen line contains 320 pels, dividing 960 by 320 gives us three dots per pel available. A little experimentation or computation shows that using a vertical dimension of two gives an image result that approximates screen proportion. Our mapping function, then, will map a pel to a 3x2 dot matrix. This also permits control of the dot density in this matrix to simulate color. Mapping for 100% density gives a very dark pel, while mapping for 30% density gives a light pel.

Now we can rough out the mapping function required. It must select pels from the regen buffer, by unpacking. These pels must be mapped to the corresponding 3x2 position in the printer buffer (left to right). The mapping function must also determine the row number of the pel and select from either the first bank of the buffer for even-numbered rows or the second bank for odd-numbered rows.

Program Description

Having determined the method of mapping, let's look at one program (see listing on page 38) that uses this scheme. PRTGRPH is a small, totally self-contained utility that will perform this mapping. It is written to produce a ".COM" file, since such a file is easier to patch if a change in control streams is required. The constants and data areas are given at the start of the program, but we'll skip over all of them initially.

The program proper starts with the control procedure, PRTGRPH. Initialization consists of setting the extra segment register to the segment address of the regen buffer (0B800H), and using standard DOS functions to prompt for (and accept) a title. The title is then "terminated" by a "$" and left in the title save area, LBUF, for later use. The printer is initialized by output of the control stream, LSPC. For the standard GRAF-

TRAX PLUS ROM, the printer control string:

Escape, A, 8

sets the printer to a line height of 8/72 inches. If you're using a different ROM, this control stream must be modified to match your ROM. A little experimentation using DEBUG to modify the control string should allow you to determine the proper line spacing for your printer.

Initialization is completed by setting the line count register (CX) to 50 (twice the lines on the screen), and the regen buffer register to a zero offset (SI). The main process loop at GRP1 is then entered. The call to PUTLNE will cause one print line to be generated and printed. The buffer register is stepped to the offset (start) of the next four display lines to be printed, and the process loops when the line count is zero. The remainder of the code in PRTGRPH simply restores the printer to normal spacing and mode, prints the saved title, and restores the printer page to the start of the next page.

The next procedure, PUTPRT, simply puts a string (terminated by a "$") to the printer. The code makes the function obvious, and I won't waste space describing this procedure.

PUTLINE clears the print buffer, builds the dot image, and then prints it. It's called once per line and is inserted in the program flow to preserve the entry values of the line counter and the regen buffer register. Useful work is performed by CLRBUF, BLDLNE, and PRTLNE which are called by this procedure.

CLRBUF again is a straightforward routine that simply pre-sets the printer buffer to binary zeros. PRTLNE simply prints the buffer, but in this case, it initializes the printer for dot matrix printing. This is done by output of the string BGRPH, which for the utilized ROM is:

Escape, L, 192, 3

This cryptic string tells the printer to enter high-density ("L") print mode for a line length of 192 + (3 * 256), or 960 dots. This re-initialization is required for each line, since the printer simply enters and stays in dot matrix mode for the byte count defined by this line length. It then returns to the mode it was in when it started the process.

BLDLNE is the basic mapping function. It begins by setting the print buffer register (DI) to zero, and an internal loop counter (BX) to the number of bytes in the line (80). A two-bit mask (0C0H) is set in the DH register, and the inner loop counter (CH is set in the outer loop, OLP. The inner loop, ILP, begins by clearing the shift count (CL) and loading the first regen byte for this print line into the AL register. This byte is masked by the content of DH and the result tested for zero. If zero, this pel is background color and is skipped. If non-zero, the call to SAVPEL is executed. In either case, the logic flow picks up at PEL2 which steps the shift count by 2 and loads the next regen data byte.

SAVPEL, which will be discussed in more detail later, simply fills in the 3x2 matrix with a dot pattern suitable for the color code determined by this masking process. Since the print line image is being built a pel at a time, left to right, the code at PEL2, PEL3, and PEL4 loads data from the appropriate bank of the regen buffer, and at the appropriate offset. Consider the first use of the routine. The SI register contains zero on entry, so the regen data load at ILP is of the first byte of the regen buffer, which corresponds to the first three dots in the first two rows of the print line (3x2 matrix again). The load at PEL2 is for the second two rows of the dot matrix print line, which correspond to the second row, first byte of the displayed image. It therefore is an odd-numbered row and must load from the second bank of the regen buffer (SI+2000H). The load at PEL3 is back to an even-numbered row, but it is now the third row of the displayed image (row 2 when considering zero as the first row), and must load from the first bank, second line, or offset 50H (80 characters into the buffer). The load at PEL4 follows the same logic, but it is again an odd-numbered row so the loading offset is 2050H.

Reaching STP, the code has completed the first three columns of the print line (eight rows), so the print buffer register

Z80 Software

SOFTWARE DESCRIPTIONS

TPM (TPM I) - $80 A Z80 only operating system which is capable of running CP/M programs. Includes many features not found in CP/M such as independent disk directory partitioning for up to 255 user partitions, space, time and version commands, date and time, create FCB, chain program, direct disk I/O, abbreviated commands and more! Available for North Star (either single or double density), TRS—80 Model I (offset 4200H) or II, Versafloppy I, or Tarbell I.

TPM-II - $125 An expanded version of TPM which is fully CP/M 2.2 compatible but still retains the extra features our customers have come to depend on. This version is super FAST. Extended density capability allows over 600K per side on an 8" disk. Available preconfigured for Versafloppy II (8" or 5"), Epson QX-10, Osborne II or TRS-80 Model II.

CONFIGURATOR I
This package provides all the necessary programs for customizing TPM for a floppy controller which we do not support. We suggest ordering this on single density (8SD).
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), LINKER ($80), DEBUG I ($80), QED ($150), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$815 Value **NOW $250**

CONFIGURATOR II
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QSAL ($200), QED ($150), ZTEL ($80), TOP II ($100), BUSINESS BASIC ($200) and MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1485 Value **NOW $400**

MODEL I PROGRAMMER
This package is only for the TRS-80 Model I. Note: These are the ONLY CDL programs available for the Model I. It includes: TPM I ($80), BUSINESS BASIC ($200), MACRO I ($80), DEBUG I ($80), ZDDT ($40), ZTEL ($80), TOP I ($80) and MODEM ($40)
$680 Value **NOW $175**

MODEL II PROGRAMMER
This package is only for the TRS-80 Model II. It includes: TPM-II ($125), BUSINESS BASIC ($200), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QED ($150), ZTEL ($80), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM ($40), MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1445 Value **NOW $375**

BASIC I - $50, a 12K+ basic interpreter with 7 digit precision.

BASIC II - $100. A 12 digit precision version of Basic I.

BUSINESS BASIC - $200. A full disk extended basic with random or sequential disk file handling and 12 digit precision (even for TRIG functions). Also includes PRIVACY command to protect source code, fixed and variable record lengths, simultaneous access to multiple disk files, global editing, and more!

ACCOUNTING PACKAGE - $300. Written in Business Basic. Includes General Ledger, Accounts Receivable/Payable, and Payroll. Set up for Hazeltine 1500 terminal. Minor modifications needed for other terminals. Provided in unprotected source form.

MACRO I - $80. A Z80/8080 assembler which uses CDL/TDL mnemonics. Handles MACROs and generates relocateable code. Includes 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical ops, local and global symbols, linkable module generation, and more!

MACRO II - $100. An improved version of Macro I with expanded linking capabilities and more listing options. Also internal code has been greatly improved for faster more reliable operation.

MACRO III - $150. An enchanced version of Macro II. Internal buffers have been increased to achieve a significant improvement in speed of assembly. Additional features include line numbers, cross reference, compressed PRN files, form feeds, page parity, additional pseudo-ops, internal setting of time and date, and expanded assembly-time data entry.

DEVELOPER I
Includes: MACRO I ($80), DEBUG I ($80), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$440 Value **NOW $150**

DEVELOPER II
Includes: MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), BUSINESS BASIC ($200), QED ($150), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM SOURCE ($40), ZTEL ($80), and DISASSEMBLER ($80).
$1280 Value **NOW $350**

DEVELOPER III
Includes: QSAL ($200), QED ($150), BUSINESS BASIC ($200), ZTEL ($80) and TOP II ($100)
$730 Value **NOW $300**

COMBO
Includes: DEVELOPER II ($1280), ACCOUNTING PACKAGE ($300), QSAL ($200) and 6502X ($150)
$1930 Value **NOW $500**

LINKER - $80. A linking loader for handling the linkable modules created by the above assemblers.

DEBUG I - $80. A tool for debugging Z80 or 8080 code. Disassembles to CDL/TDL mnemonics compatible with above assemblers. Traces code even through ROM. Commands include Calculate, Display, Examine, Fill, Goto, List, Mode, Open File, Put, Set Wait, Trace, and Search.

DEBUG II - $100. A superset of Debug I. Adds Instruction Interpreter, Radix change, Set Trap/Conditional display, Trace options, and Zap FCB.

6502X - $150. A 6502 cross assembler. Runs on the Z80 but assembles 6502 instructions into 6502 object code! Similar features as our Macro assemblers.

QSAL - $200. A SUPER FAST Z80 assembler. Up to 10 times faster than conventional assemblers. Directly generates code into memory in one pass but also to offset for execution in its own memory space. Pascal like structures: repeat...until, if...then...else, while...do, begin...end, case...of. Multiple statements per line, special register handling expressions, long symbol names, auto and modular assembly, and more! This one uses ZILOG Mnemonics.

QED - $150. A screen editor which is both FAST and easy to learn. Commands include block delete, copy, and move to a named file or within text, repeat previous command, change, locate, find at start of line, and numerous cursor and window movement functions. Works with any CRT having clear screen, addressable cursor, clear to end of line, clear to end of screen, and 80X24.

ZTEL - $80. An extensive text editing language and editor modelled after DEC's TECO.

ZEDIT - $50. A mini text editor. Character/line oriented. Works well with hardcopy terminals and is easy to use. Includes macro command capability.

TOP I - $80. A Text Output Processor for formatting manuals, documents, etc. Interprets commands which are entered into the text by an editor. Commands include justify, page number, heading, subheading, centering, and more.

TOP II - $100. A superset of TOP I. Adds: embedded control characters in the file, page at a time printing, selected portion printing, include/merge files, form feed/CRLF option for paging, instant start up, and final page ejection.

ZDDT - $40. This is the disk version of our famous Zapple monitor. It will also load hex and relocatable files.

ZAPPLE SOURCE - $80. This is the source to the SMB ROM version of our famous Zapple monitor. It can be used to create your own custom version or as an example of the features of our assemblers. Must be assembled using one of our assemblers.

MODEM - A communication program for file transfer between systems or using a system as a terminal. Based on the user group version but modified to work with our SMB board or TRS-80 Models I or II. You must specify which version you want.

MODEM SOURCE - $40. For making your own custom version. Requires one of our Macro Assemblers.

DISASSEMBLER - $80. Does bulk disassembly of object files creating source files which can be assembled by one of our assemblers.

HARDWARE

S-100 — SMB II Bare Board $50. "System Monitor Board" for S-100 systems. 2 serial ports, 2 parallel ports, cassette interface, 4K memory (ROM, 2708 EPROM, 2114 RAM), and power on jump. When used with Zapple ROM below, it makes putting a S-100 system together a snap.
Zapple ROM $35. Properly initializes SMB I/II hardware, provides a powerful debug monitor.

IBM PC — Big Blue Z80 board $595. Add Z80 capability to your IBM Personal Computer. Runs CP/M programs but does not require CP/M or TPM. Complete with Z80 CPU, 64K add on memory, serial port, parallel port, time and date clock with battery backup, hard disk interface, and software to attach to PC DOS and transfer programs. Mfr'd by QCS.
50% Discount on all CDL software ordered at the same time as a Big Blue (and for the Big Blue).

APPLE II — Chairman Z80 $345. Add Z80 capability to your Apple II/II Plus computer. Runs CP/M programs with our more powerful TPM. Includes 64K memory add on (unlike the competition this is also useable by the 6502/DOS as well as the Z80), TPM, QSAL assembler, QED Screen Editor, and Business Basic. Mfr'd by AMT Research.
Apple Special $175. Buy the Apple Z80 Developer at the same time as the "Chairman" and pay only $175 instead of $325.

APPLE Z80 DEVELOPER
Includes: 6502X ($150), MACRO II ($100), MACRO III ($150), QSAL ($200), QED ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), ZDDT ($40) and BUSINESS BASIC ($200)
VALUE: $1250 **NOW $325**
$175 when purchased with AMT "Chairman" Board

DISK FORMATS

When ordering software specify which disk format you would like.

CODE	DESCRIPTION
8SD	8" IBM 3740 Single Density (128 bytes/26 sectors/77 tracks)
8DD	8" Double Density (256 bytes/26 sectors/77 tracks)
8XD	8" CDL Extended Density (1024 bytes/8 sector/77 traceks = 616K)
5SD	5.25" Single Density (TRS80 Model I, Versafloppy I, Tarbell I)
5EP	5.25" Epson Double Density
5PC	5.25" IBM PC Double Density
5XE	5.25" Xerox 820 Single Density
5OS	5.25" Osborne Single Density
5ZA	5.25" Z80 Apple (Softcard compatible)

TPM INFO When ordering TPM I or II, in addition to Disk Format, please specify one of the following codes:

	CODE	DESCRIPTION
TPM I:	NSSD/H	North Star Single Density for Horizon I/O
	NSSD/Z	North Star Single Density for Zapple I/O
	NSDD/H	North Star Double Density for Horizon I/O
	NSDD/Z	North Star Double Density for Zapple I/O
	TRS80-I	TRS-80 Model I (4200H Offset)
	TRS80II	TRS-80 Model II
	VI8	Versafloppy I 8"
	VI5	Versafloppy I 5.25"
TPM-II:	VII8	Versafloppy II 8" (XD)
	VII5	Versafloppy II 5.25"
	TRS80II	TRS-80 Model II (XD)

Prices and Specifications subject to change without notice.
TPM, Z80, CP/M, TRS80 are trademarks of CDL, Zilog, DRI and Tandy respectively.

ORDERING INFORMATION:

VISA/MasterCard/C.O.D.
Call or Write With Ordering Information....

OEMS:

Many CDL products are available for licensing to OEM's. Write to Carl Galletti with your requirements.

Dealer Inquiries Invited.

For Phone Orders ONLY Call Toll Free...
1-(800) 458-3491 (Except Pa.)
Ask For Extension #15

For information and Tech Queries call
(609) 599-2146

Computer Design Labs

342 Columbus Avenue/Trenton, NJ 08629

Circle no. 13 on reader service card.

is stepped by three. The mask is shifted to the next pel position, and the inner loop count decremented. This inner loop will execute four times (for the four pels in the screen byte), and then exit to a step of the regen buffer register, moving to the next four pels. The outer loop counter (BX is decremented, and the process continues until 80 regen buffer bytes have been processed. This corresponds to one display line, and the build function exits.

To complete the code description, SAVPEL simply converts the color code masked from the pel to a dot matrix code of suitable density and stores this code in the print line buffer. You'll note that the mask saved in the print line buffer is saved in a two-bit space (the two part of the 3x2 matrix) and is saved in three bytes (the three part). Since we don't know in which two-bit field the pel is located in this procedure, the test checks all four two-bit fields. It checks first for the presence of a "one bit" in the two-bit fields. If not found, it assumes that the color code is a two, since the routine would not be entered if the color code is zero. If a one bit is found, then a two bit is checked. If not found, the color code is one, while if found, the color code is three. The DL mask and the CL shift may be modified to give different shade patterns. The one provided in the code seems to give a good contrast between colors, yet provides a pleasing overall density. You can try other combinations and select the one most suited to your printer (and your eye).

That's it. The listing is relatively well commented, so try reading through the code if this description doesn't seem clear.

Assembly and Link

Key in the source code as shown in the listing, and assemble using either of the assemblers. If you have a different printer or control ROM, you should change the printer control strings to match your configuration. Link as you would any other program. The link will respond with a "warning" message informing you that there is no stack segment and will give an error count of one. Ignore this error message and execute EXE2BIN. Rename the resultant file "name.COM" and execute.

For those of you who are operating at PC-DOS v.1.0, you may not have the DOS utility EXE2BIN, which is almost necessary to produce a ".COM" file. You

Kaliedoscope

Figure 1

may, therefore, prefer an ".EXE" file. If so, you must add a stack segment, initialize your data segment register, initialize the stack for the eventual return to DOS, and change the PRTGRPH procedure to a "FAR."

Use of the Utility

Use of the utility is primitive, with one big "hooker." The utility will print the content of the color/graphics regeneration buffer, *at the instant of execution.* How you invoke this utility without disturbing this buffer is a function of how you're configured. I'm working with both the monochrome and the color/display adapters, so it's easy for me. I simply arrange my graphics program so that it does *not* clear the graphics screen on exit, and switches back to the monochrome display before exit. That leaves the graphics regen buffer containing the image I want, and I can enter the command to execute this utility on the monochrome display. If you have only the color/graphics adapter, you'll have to be considerably more complex in your process. One trick is to include code in your display generation program that writes the entire regen buffer to disk under some condition. This utility can then be simply modified to load the buffer image into memory and point to it instead of the regen buffer.

Finally, this program is hereby placed in public domain. You may copy it, utilize it in your own work, etc. freely and without notification.

Although this utility is small and not too much keying is required, I will provide copy service. A single-surface diskette and a check for $5 will get you a copy of the program source materials, object file, and an executable ".COM" file. A check for $7.50 will get you a new Verbatim (or equivalent) containing the same materials. I'll furnish the return mailer and will ship UPS Blue Label unless otherwise instructed. Send to: Dan Daetwyler, Route 5, Box 518A, Springdale, AR 72764 (that's Arkansas, not Arizona).

If you have trouble/questions with this one, you can write me at the above address, or call 501-756-0212. (I'm usually around 24 hours, seven days.) Good luck and happy printing.

DDJ

(Listing begins on page 38)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 235**

Printing Graphics on the IBM PC (Text begins on page 32)

```
 COMMENT \
********************************************************************************
*                    Copyright 1983 - DD Systems - Springdale, AR              *
********************************************************************************
*                                                                              *
*         Output Color Graphic Screen to Printer                              *
*                                                                              *
*                                                                              *
*         This self-contained utility maps the "pel" maps of the color graphic *
*         refresh buffer to the printer.  It assumes the image has been        *
*         loaded by other services.                                            *
*                                                                              *
********************************************************************************\

;
CODE     SEGMENT PARA PUBLIC 'CODE'
         ASSUME  CS:CODE,DS:CODE
         ORG     100H
BEGIN:   JMP     PRTGRPH
;
PRMPT    DB      'Enter Title: $'              ;Prompt for title line
LSPC     DB      27,'A',8,13,10,'$'            ;Sets printer to 8/72" spacing
BGRPH    DB      27,'L',192,3,'$'              ;Sets to hi-density dot graphics
NORM     DB      27,'@','$'                    ;Resets to normal mode
CRLF     DB      13,10,'$'
;
LBUF     DB      80
         DB      82 DUP (?)                    ;Title buffer
PBUF     DB      960 DUP (?)                   ;Bit graphics buffer
         DB      '$'


;
PRTGRPH PROC     NEAR
         MOV     AX,0B800H
         MOV     ES,AX                         ;Addressability of regen buffer
         MOV     DX,OFFSET PRMPT
         MOV     AH,9
         INT     21H                           ;Prompt for caption
         MOV     DX,OFFSET LBUF
         MOV     AH,10
         INT     21H                           ;Get title line
         MOV     AL,LBUF+1                     ;Get entry length
         CBW
         MOV     SI,AX
         MOV     LBUF+2[SI],'$'                ;Set line terminator
         MOV     SI,OFFSET LSPC
         CALL    PUTPRT                        ;Set printer
         MOV     CX,50                         ;Print line count
         MOV     SI,0                          ;Start of regen buffer
GRP1:    CALL    PUTLNE                        ;Output one line
         ADD     SI,160                        ;  which is four screen rows
         LOOP    GRP1
         MOV     SI,OFFSET NORM
         CALL    PUTPRT                        ;Restore printer to normal
         MOV     SI,OFFSET CRLF
         CALL    PUTPRT                        ;Space line
         MOV     SI,OFFSET LBUF+2
         CALL    PUTPRT                        ;Output title
         MOV     CX,31
GRP2:    MOV     SI,OFFSET CRLF
         CALL    PUTPRT                        ;Space to end of page
         LOOP    GRP2
         MOV     SI,OFFSET NORM
         CALL    PUTPRT
         RET
```

```
PRTGRPH ENDP
;
;        Simple "put string" to printer
;
PUTPRT  PROC    NEAR
        PUSH    SI
        PUSH    DX
PPLP:   MOV     DL,BYTE PTR[SI]
        CMP     DL,'$'                  ;Check for "end of string" mark
        JZ      PPDNE
        MOV     AH,5
        INT     21H                     ;DOS Printer call
        INC     SI
        JMP     PPLP
PPDNE:  POP     DX
        POP     SI
        RET
PUTPRT  ENDP
;
;        Output one graphic "line"
;
PUTLNE  PROC    NEAR
        PUSH    SI
        PUSH    CX
        CALL    CLRBUF                  ;Clear bit buffer
        CALL    BLDLNE                  ;Build dot image
        CALL    PRTLNE                  ;Output line
        POP     CX
        POP     SI
        RET
PUTLNE  ENDP
;
;        Clear line buffer to null
;
CLRBUF  PROC    NEAR
        XOR     DI,DI
        MOV     AX,DI
        MOV     CX,480
CBLP:   MOV     WORD PTR PBUF[DI],AX    ;Clears line buffer to null
        INC     DI
        INC     DI
        LOOP    CBLP
        RET
CLRBUF  ENDP
;
PRTLNE  PROC    NEAR
        MOV     SI,OFFSET BGRPH
        CALL    PUTPRT                  ;Set printer for dot graphic line
        MOV     SI,OFFSET PBUF
        MOV     CX,960                  ;  and output bit stream
PL1:    MOV     DL,BYTE PTR [SI]
        MOV     AH,5
        INT     21H
        INC     SI
        LOOP    PL1
        RET
PRTLNE  ENDP
;
;        Build dot graphic bit stream in buffer
;
BLDLNE  PROC    NEAR
        XOR     DI,DI                   ;Dot buffer index
        MOV     BX,80                   ;Loop count for line
OLP:    MOV     DH,0C0H                 ;Outer loop mask
        MOV     CH,4                    ;Inner loop counter
ILP:    XOR     CL,CL                   ;Shift for pel save
```

(Continued on page 41)

Overbeek Enterprises

OVERBEEK ENTERPRISES is rapidly establishing a broad selection of inexpensive CP/M programs. We will be adding new selections continuously– as fast as we can bring them into the market, while making absolutely sure that each one is a real bargain. Substantial quantity discounts are available. Our average time to process an order is 2 days. If for any reason you cannot make one of our products run on your system, we will refund the purchase price.

$29.95 Menu-Plus — Menu System

You've probably heard about the glories of menu driven systems. This powerful package developed by Capacity Inc. makes the ease of menu driven systems affordable to any CP/M user. Menu-Plus allows rapid, easy configuration of simple or hierarchical menus. You can hide the complexities of CP/M and allow single keystroke invocation of your programs. It's a user's dream. Both beginners and experienced operators will find Menu-Plus a significant enhancement. You do not have to be a programmer to install or tailor Menu-Plus on your system. In just a few minutes, you can easily create whatever new menus you desire with a text editor. Our competitors offer products in the $75-$150 range. We invite comparison of this product with any other menu system in terms of features or user friendliness. In terms of price there is no comparison.

$29.95 WSMX80 — Wordstar/Epson Print Processor

Can you print these on your Epson?

$$R = \frac{e^{-2\alpha t}}{(2\pi sin\theta)^2} \times t_3/\tau \qquad M = \int_{t-0}^{\infty} a \, d\rho$$

WSMX80 has been created for WordStar users that have an Epson MX-80 or MX-100 printer equipped with either the Graftrax or the Graftrax Plus option. By separating the printing function from WordStar, all of the features of the printer can be used to full effect. Now you can use alternate character sets, compressed fonts, italics, and a variety of other features based on the Graftrax capabilities. WSMX80 has been widely used at the University of Kansas for over a year and has proved to be an invaluable tool.

$29.95 Micro-WYL — Text Editor

Tired of trying to use ED under CP/M? Here are just a few unsolicited quotes from our customers:

"I operate a software house in Central California and have used a lot of text editors over the years. Micro-WYL has to be one of the best I have ever used, regardless of price."

"Micro-WYL is undoubtedly the hottest bargain on the market."

"Thank you, thank you, thank you."

"This editor is perfect for writing in nearly any programming language. [I] . . . have no hesitation in recommending it to anyone whose requirements match the capabilities of this inventive piece of software." *–From a review in Infoworld (11/15/82)*

Now you can have the convenience of WYLBUR on any Z80 CP/M system.

$29.95 Disk Inspector — Disk Editor

Have you ever been unable to read a file due to a bad sector? Have you ever erased the wrong file? Disk Inspector acts as a full-screen editor for diskettes. You can simply watch as sectors are displayed on the screen in both character and hex formats. When you wish to make the display pause, touch the spacebar. If you wish to alter a sector, it is a simple matter to move the cursor over the appropriate character, alter it, and have the sector rewritten.

Although Disk Inspector runs only on Z80 CP/M systems, you can inspect and alter normal (non CP/M) Apple diskettes, as well. The disk drives may be single or double density, single or double sided.

Note: Disk Inspector requires an 80x24 screen on your CRT.

CP/M is a registered trademark of Digital Research, Inc.

WYLBUR is a registered trademark of The Board of Trustees of the Leland Stanford Junior University

WordStar is a registered trademark of MicroPro International

- ☐ 8" SSSD
- ☐ Apple/Softcard
- ☐ KAYPRO II 5"
- ☐ NEC 5"
- ☐ Zerox 820
- ☐ ALTOS Series 5
- ☐ Televideo TS-802
- ☐ Osborne
- ☐ Superbrain
- ☐ Heath/Magnolia

Northstar 5" DD
- ☐ Advantage
- ☐ Horizon
- ☐ Morrow Micro Decision

Amount:

$29.95 for Menu Plus ________ $29.95 for Disk Inspector________

$29.95 for WSMX80 ________ $2 for postage & handling________

$29.95 for Micro-WYL ________ **TOTAL** ________

Name________________________________

Address______________________________

City__________________ State________ Zip________

Make your check payable to: **Overbeek Enterprises**
P.O. Box 726
Elgin, IL 60120

To order C.O.D. or with a MasterCard or Visa call **312-697-8420** between 9 am and 5 pm (CST)

"They want how much?!"

Circle **no. 59** on reader service card.

```
        MOV     AL,BYTE PTR ES:[SI]      ;Get screen byte
        AND     AL,DH                    ;Mask
        JZ      PEL2
        CALL    SAVPEL                   ;non-zero, so print blob
PEL2:   ADD     CL,2
        MOV     AL,BYTE PTR ES:[SI+2000H] ;Next row
        AND     AL,DH
        JZ      PEL3
        CALL    SAVPEL
PEL3:   ADD     CL,2                     ;Stepping shift count
        MOV     AL,BYTE PTR ES:[SI+50H]  ;Row three
        AND     AL,DH
        JZ      PEL4
        CALL    SAVPEL
PEL4:   ADD     CL,2
        MOV     AL,BYTE PTR ES:[SI+2050H] ;Fourth row
        AND     AL,DH
        JZ      STP
        CALL    SAVPEL
STP:    ADD     DI,3                     ;Step save buffer pointer
        SHR     DH,1
        SHR     DH,1                     ;Reposition pel mask
        DEC     CH
        JNZ     ILP                      ;Continue inner loop for next col
        INC     SI                       ;Step regen buffer ptr
        DEC     BX
        JNZ     OLP                      ;Outer loop
        RET
BLDLNE  ENDP
;
;       Save bit image for this pel - different "shade" for each color
;
SAVPEL  PROC    NEAR
        TEST    AL,55H                   ;One bit on
        JNZ     ONEB
        JMP     DOTWO
ONEB:   TEST    AL,0AAH                  ;Test for two bit
        JNZ     DOTHRE
;   The following is for a data bit of 01
        MOV     DL,0C0H                  ;Load mask
        SHR     DL,CL                    ;  and position
        OR      PBUF[DI],DL
        OR      PBUF[DI+1],DL
        OR      PBUF[DI+2],DL
        RET
DOTWO:  MOV     DL,80H
        SHR     DL,CL
        OR      PBUF[DI],DL
        OR      PBUF[DI+2],DL
        SHR     DL,1
        OR      PBUF[DI+1],DL
        RET
DOTHRE: MOV     DL,0C0H
        SHR     DL,CL
        OR      PBUF[DI+2],DL
        RET
SAVPEL  ENDP
;
CODE    ENDS
;
        END     BEGIN
```

End Listing

The Game of Life

on the IBM-PC

Computer programs for running John Conway's Game of Life were quite popular a few years ago. The problem with these older simulations was usually their speed: a single generation on a ten-by-ten grid could take up to 90 seconds. Presented here is a version of the game for the IBM/PC computer. Features of this version are adjustable speed (with 2.7 generations/sec as a top speed), easy entry of the seed generation (via a screen-editor), and marking of cells due to be "born," and cells remaining alive each generation (as different from cells marked to die). This feature gives a better impression of fluidity on the grid.

This version requires only the IBM monochrome display to run, and will run under either MS-DOS or CP/M-86, since only calls to the IBM ROM are made.

Background

The Game of Life takes place on a square grid. Every cell on the grid can be either alive or dead. Every cell on the grid has eight neighbors. Each generation, every cell on the grid is evaluated. The cell will remain alive if it has two or three neighbors, or will die if it has less than two (from loneliness), or more than three (from overpopulation). An empty cell will have a "birth" (a live cell will be placed there the next generation) if it has exactly three neighbors. The game was developed by John Conway in 1976.

by Simson L. Garfinkel

Simson L. Garfinkel, 18 Dartmouth Lane, Haverford, Pennsylvania 19041.

The Program

The program is straightforward. When first run, the screen is cleared and the cursor is positioned in the center. The cursor may be moved by pressing the four arrow keys on the keyboard (Num Lock doesn't matter because the keyscan code is interrogated, not the ASCII code). A live cell may be deposited by depressing Ins, and a cell may be cleared by pressing Del. When the screen is complete, pressing Esc starts evaluation of the next generation.

The program stores the grid in the screen memory of the monochrome display. Associated with every displayable character on the screen are two bytes of memory, the low byte for the character displayed, and the hi byte for the display attributes (underlined, reversed, etc.). The program uses the attribute byte of every screen position as a second array to hold the next generation in while it evaluates the present generation.

After the time delay, subroutine *count* is called. *Count* counts the number of neighbors that every cell has, and decides whether or not the cell will be alive in the next generation. If it is going to be alive, the display attribute for that character is set to *rev*. If the cell is going to be dead, the attribute is set to *dark*. *Rev* and *dark* are reverse video and normal video in my listing, so when a cell is going to have a birth or stay alive, it is inverted on the screen, but *rev* and *dark* can be changed to three and five, causing both to display as normal if the flickering this produces is annoying.

After all of the decisions regarding life and death are made, subroutine *update* goes through screen memory putting in the character for a live cell if the cell is supposed to be alive, or a dead cell if the cell is supposed to be dead, and resetting the attribute byte.

The program then looks for a user key press. If one has occurred, the program quits if it was an Esc, or changes the time delay value if it was a digit. Pressing 0 gives no delay, or about 2.7 generations per second. Pressing 9 gives a 3.5 second delay per generation.

Expansions

The one problem with this implementation is that the screen doesn't have enough rows to allow a simulation of a complex colony. I would love to see an implementation of life in medium-resolution color graphics, with births in blue and deaths in red, but I don't have access to a color display, so that will have to wait. The program could also be cleaned up to make it run faster, but 2.7 generations a second is really fast enough for most applications. DDJ

(Listing begins at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 237**

Game of Life (Text begins on page 42)

```
Comment $

        ********************************
        *                              *
        *       The Game of Life       *
        *                              *
        ********************************

        John Conway's mathematical game of life, implemented on
        the IBM/PC, by Simson L. Garfinkel.

        Written in 8088 assembly language using the Microsoft
        Macro Assembler.

        Notes on running the program:

          When program is run:

                1.  Screen clears.
                2.  User enters first generation from keyboard.
                    Arrow keys move the cursor.  INS key deposits
                    a live cell, DEL removes a live cell, (in case
                    the user makes a mistake.)
                3.  Pressing ESC starts program.
                4.  For each generation, cells which will have life
                    on the next turn are inverted.
                5.  Screen is updated to next generation.
                6.  Keyboard is interrogated for command.
                7.  If ESC is pressed, program terminates.
                8.  If a number 0-9 is pressed, speed is selected.
                    At speed 0, approx. 2.7 generations/sec are performed
                    At speed 9, each generation takes 3.5 sec.
                9.  program loops to #4.

        $

        ;global definitions

live    equ  02             ;character for a live cell
dead    equ  00             ;character for a dead cell

rev     equ  70h            ;reverse video (marks cell to live)
dark    equ   2             ;normal video  (marks cell to die )

time    equ  300            ;time delay base

TERM    equ  27             ;Character to exit mode


cseg    segment para public 'code1'
start   proc far
        assume cs:cseg,ds:nothing,ss:stack,es:nothing

                            ;set up return location
```

(Continued on next page)

```
        push ds
        sub  ax,ax
        push ax            ;now I can go home when I'm finished.


        call Enter         ;Enter board
        mov  cx,0          ;initial delay, 0
main:
        push cx            ;save delay variable
        cmp  cx,0
        jz   s13

s1:     push cx
          mov  cx,time
s11:      push cx
            mov  cx,time
s12:        loop s12
          pop  cx
          loop s11
        pop  cx
        loop s1            ;what a time delay!

s13:    call count         ;Count up every cell's neighbours,
        call update        ;Update screen
        pop  cx            ;get back the time delay

        mov  ah,1          ;See if user has pushed a key
        int  16h
        jz   main          ;nope - loop back

        mov  ah,0          ;get the character our of the buffer
        int  16h

        cmp  al,TERM
        jnz  s2
        ret                ;finished - go back to ms/dos

s2:     cmp  al,'0'        ;see if it is a speed command
        jb   main
        cmp  al,'9'
        jnbe main
                           ;It's a number
        sub  al,'0'        ;now it goes from 0 to 9
        mov  ah,0
        mov  cx,ax         ;put it in cx
        jmp  main
start   endp



Enter   proc near          ;Subroutine to enter board
                           ;define scan-codes:
left    equ  75
right   equ  77
up      equ  72
```

```
down     equ  80
point    equ  82
del      equ  83
esc      equ   1

         call cls         ;clear the screen

         ;Registers are used as follows:
         ;DH - Y position
         ;DL - X position

         mov  dh,12
         mov  dl,40

e1:      mov  bh,0        ;move the cursor to x,y position
         mov  ah,2        ;code for cursor move
         int  10h         ;interrupt for cursor move

         mov  ah,0        ;set up to read the next keypress
         int  16h         ;keypress read

         cmp  ah,left     ;make a rational decision about the users
         jz   go_left     ;entry.
         cmp  ah,right
         jz   go_right
         cmp  ah,up
         jz   go_up
         cmp  ah,down
         jz   go_down
         cmp  ah,point
         jz   go_point
         cmp  ah,del
         jz   go_del

         cmp  ah,esc
         jnz  e1          ;loop back - unknown command
         mov  dx,23*256   ;put the cursor at lower-left hand corner
         mov  ah,2
         int  10h
         ret              ;go back to caller

go_left:                  ;move left if I can
         cmp  dl,0        ;in leftmost collumn?
         jz   e1          ;yes - go back
         sub  dl,1        ;no  - subtract one
         jmp  e1          ;go back

go_right:                 ;move right if I can.
         cmp  dl,79
         jz   e1
         add  dl,1
         jmp  e1

go_up:                    ;go up if I can
         cmp  dh,0
         jz   e1
         sub  dh,1
         jmp  e1
```

(Continued on next page)

Game of Life (Listing continued, text begins on page 42)

```
go_down:                    ;go down if I can
        cmp   dh,24
        jz    e1
        add   dh,1
        jmp   e1

go_point:                   ;put a live dot where the cursor is -- don't move it
        mov   al,live       ;it's the live character
gp2:    mov   cx,1          ;one character to write
        mov   ah,10         ;code to write character
        int   10h           ;do it
        jmp   e1            ;get next command

go_del:                     ;delete character at cursor
        mov   al,dead
        jmp   gp2           ;let go_point do the rest
Enter   endp



Cls     proc near           ;Subroutine to clear the screen
        mov   ax,6*256
        mov   cx,0
        mov   dx,24*256+79
        mov   bh,2
        int   10h
        ret
cls     endp


Count   proc near           ;Subroutine to count up every cell's neighbours
        ;Registers used:
        ;DH,DL:  Y,X of current cell being interrogated
        ;DS    :  Base offset - into screen memory
        ;DI    :  offset for character presently being looked at
        ;
        ;Outline for each character
        ;   1.  Count up number of neighbours
        ;   2.  If three neighbours, or if two and cell is live, put
        ;       a rev  on the screen at the attribute position, else
        ;       put a dark
        ;   3.  Go to next character


chk     macro   yy,xx
        local   ch1,offs
offs    equ     (xx+yy*80)*2
        mov     cx,[di+offs]            ;get byte to check
        cmp     cl,live                 ;check to see if this cell is alive
        jnz     ch1                     ;nope
        add     al,1                    ;yes - increase neighbour count
ch1:
        endm
```

(Continued on page 50)

Now for Concurrent CP/M-86

VEDIT - The Clear Choice for Programmers

Plus Features for Fast & Efficient Word Processing

Increasing your productivity is what a good text editor is all about. VEDIT excels by giving you a unique combination of extensive and easy to use editing features, customizability and complete hardware support. So compare VEDIT. You'll find everything you expect in a good editor plus a variety of time saving features which only VEDIT offers.

VEDIT is fully user oriented. You can use the function keys on any keyboard, or a layout you are already familiar with - simplifying your usage and easing your learning. While most editors lose text if you run out of disk space, VEDIT lets you delete files or change disks. VEDIT is the result of continuous enhancement and feedback from our nearly 4000 users.

VEDIT
CP/M
CUSTOMIZABLE
FULL SCREEN EDITOR

For program development it surpasses any other editor - with more extensive file handling, important command macro capability and special features for Pascal, PL/1, 'C', Cobol, Assembler and others. With VEDIT you will reduce your program editing time by 30% as compared to the best word processor.

For word processing, VEDIT has word wrap, adjustable margins, reformatting of paragraphs, word and paragraph functions and simple printing with imbedded printer control characters.

Command macros let you perform editing tasks you might otherwise not even attempt. Time consuming tasks for other editors (such as translations or extensive search/replace on many files), can be done by VEDIT without your intervention, even overnight if you choose.

VEDIT supports all of the new CRT terminals, video boards and 8080, Z80 and 8086 computers. We have been consistently first to support new computers - first for CP/M-86, first for MSDOS. And we will support you with any technical assistance you may need.

For the full story, purchase VEDIT risk free. Evaluate the 125 page manual and if you are not satisfied, return the package (disk unopened) for a courteous refund.

COMPARE VEDIT

Feature	VEDIT	WordMaster	WordStar
True Full Screen Editing	Yes	Yes	Yes
Edit files one disk in length	Yes	Yes	Yes
Compact and fast	Yes	Yes	No
Display of line and column #	Yes	No	Yes
Set/Goto text markers	Yes	No	Yes
'Undo' key to restore line	Yes	No	No
Automatic Indent/Undent	Yes	No	No
Adjustable tab positions	Yes	No	Yes
Repeat function key	Yes	Yes	No
Text move and copy	Yes	Yes	Yes
Scratchpad buffers	10	Only 1	No
Load/Save buffers on disk	Yes	No	No
Flexible command mode	Yes	Yes	No
Multiple command macros	Yes	No	No
Directory display	Yes	No	Yes
Edit additional (small) files simultaneously	Yes	No	No
Insert another disk file	Yes	Yes	Yes
Unlimited file handling	Yes	No	No
Automatic disk buffering	Yes	Yes	Yes
Recovery from 'Full Disk'	Yes	No	Some
Change disks while editing	Yes	No	No
Startup command file	Yes	No	No
Program CRT function keys	Yes	No	No
Word Wrap and reformatting	Yes	No	Yes
Printing	Simple	No	Extensive
Print formatting	No	No	Yes
Menu driven installation	Yes	No	Yes
Support newest CRT terminals	Yes	No	No
Support smart CRT functions	Yes	No	Some
Customizable keyboard layout	Yes	No	No
Available for CP/M-86	Since 1981	?	?
Available for MSDOS	Since 1981	?	Yes

Please specify your microcomputer, video board or the CRT terminal version, 8080, Z80, or 8086 code, operating system and disk format.

VISA & MasterCard

VEDIT - Disk and Manual
For 8080, Z80 or IBM PC $150
For CP/M-86 or MSDOS $195
Manual only $18

Zenith Z100 and Z89 • DEC VT180 • Televideo 802
TRS-80 I, II and 16 • Xerox 820 • Apple II Softcard
SuperBrain • NorthStar • Cromemco • Altos • Vector
MP/M • CP/M-86 • MP/M-86 • MSDOS • PCDOS

IBM Personal Computer and IBM Displaywriter

CompuView
PRODUCTS, INC.

1955 Pauline Blvd., Suite 200 • Ann Arbor, Michigan 48103 • (313) 996-1299

IN AUSTRALIA DISTRIBUTED BY SOFTWARE SOURCE PTY. LTD.
89 OXFORD ST., BONDI JUNCTION, SYDNEY - (02) 389-6388

Circle no. 16 on reader service card.

Game of Life (Listing continued, text begins on page 42)

```
        mov   ax,0B000H
        mov   ds,ax         ;offset value for monochrome display

        mov   dh,1          ;Start at 1,1 and go to 23,78
        mov   dl,1          ;to prevent wrap-arround

c1:     mov   ax,160        ;get true offset from ds into screen memory
        mul   dh

        mov   cx,dx
        mov   ch,00         ;just get dl
        add   ax,cx
        add   ax,cx         ;ax:=(dh*80+dl)*2

        mov   di,ax         ;di:=ax
        mov   ax,0          ;ax will be used for neighbour counting

        chk   -1,-1         ;count number of neighbours
        chk   -1, 0
        chk   -1,+1
        chk    0,-1
        chk    0,+1
        chk   +1,-1
        chk   +1, 0
        chk   +1,+1         ;test all of the neighbours

        mov   cx,[di]       ;get byte to check
        cmp   al,3
        jz    give_life     ;life if has 3 neighbours
        cmp   cl,live       ;is it alive?
        jnz   give_death    ;no
        cmp   al,2          ;he lives if he has 2 neighbours and he is already
                            ;alive
        jnz   give_death    ;nope

give_life:                          ;make this one alive
        mov   ch,rev
        jmp   c2

give_death:
        mov   ch,dark
c2:     mov   [di],cx               ;put back on the screen

next_cell:
        cmp   dl,78         ;am I at the end of the X line?
        jz    c3            ;yes
        add   dl,1          ;nope
        jmp   c1
c3:     mov   dl,1
        cmp   dh,23         ;am I at the end of the Y line?
        jz    c4            ;yes
        add   dh,1          ;nope
        jmp   c1
c4:     ret                 ;yes - go home!
Count   Endp
```

```
Update  proc  near                ;This updates the generation on the screen
        mov   ax,0B000H           ;Get screen offset
        mov   ds,ax

        mov   bx,24*80*2-2        ;loop through all of the screen but last line
u1:     mov   cx,[bx]             ;line
        cmp   ch,rev              ;is it to live?
        jnz   u2                  ;no
        mov   cl,live             ;yes
        jmp   u3
u2:     mov   cl,dead             ;no
u3:     mov   ch,dark             ;turn off reverse
        mov   [bx],cx             ;put it back on the screen
        sub   bx,2                ;loop back until done with the screen
        jg    u1
        ret                       ;go back to caller
Update  endp
cseg    ends


stack   segment para stack 'stack'
        db 30 dup('stack   ')
stack   ends


        end
```

End Listing

Series Expansion in Forth

Series expansions are widely used in computers to generate precision non-linear functions. The usual approaches to series implementation use a lot of constants. This results in much space being used for headers or for stuffing all the constants into arrays, which makes checking and adding more constants awkward. Also, the mechanics of computing each series varies – some have only even and others only odd exponents, some have all exponents, some begin with a constant and others do not, etc.

A different approach, which places coefficient-exponent pairs into an array while preserving readability and access, is shown in screens 38-43, at right. The algorithm is identical for all series, regardless of first-term constant or exponent pattern. This approach, like Dr. Eaker's Case statement, permits the programmer to concentrate on the results rather than on the mechanics.

The first goal is to compactly store the data, which consists of floating-point coefficients and integer exponents. SERIES-CONSTANTS expects on the stack the number of terms to be computed. It then allots space for this number (1 byte) plus 7 bytes per term – one byte for the integer exponent and 6 bytes for the floating-point coefficient. This arrangement limits the number of terms and the exponent to 256, which would seem adequate for most needs. A header is compiled for the assigned NAME and the number of terms is stored in the first byte of the allotted space.

Next, the coefficients and exponents are written in the eminently readable format shown in Screen 227 (on page 55) which puts them on the stack in the reverse of the order shown, with the smallest exponent on top. Screen 228 contains the code to obtain the proper sign and to reduce the angle to within less than 360°.

These constants and exponents are next loaded into the allotted space with the phrase "n NAME LOAD-CONSTANTS." The number of constants to be stored is checked against the number for which space was allotted, and the

by Wendall C. Gates

Wendall C. Gates, PE, Advanced Instrumentation Inc., Box 2070, Santa Cruz, California 95063.

Series Expansion

```
Screen # 38
( Series Expansion Statement by W.C. Gates   11-21-82 )
0 VAL NEXTADD      F0 FVAL FACCUM
0 VARIABLE POWER   0 VARIABLE NEXTEXP
: OVARS F0 TO FACCUM  0 POWER ! ;  ( Zero FACCUM, POWER  )

( NGET does  z add ... z add n  with add+1 in NEXTADD )
: NGET  DUP 1+ TO NEXTADD C@ ;

( n SERIES-CONSTANTS  name    defines the storage space for an)
( array which contains [number of terms] [exp. of 1st term]   )
( [FP coefficient of 1s term] [exp of 2nd term][next coeff.]  )
( exponents rise monotonically, but gaps OK   )
: SERIES-CONSTANTS
  <BUILDS DUP HERE C!    7 * 1+ ALLOT   ( for 6-byte floating- )
   DOES>  ;                             ( point numbers        )
-->


Screen # 39
( Series Expansion-- LOAD-CONSTANTS )
(                                                              )
( LOAD-CONSTANTS stores the pre-computed coefficients into the )
( space alloted by SERIES-CONSTANTS. These coefficients must be)
( on the stack already, together with the exponent for each    )
( Cn exp[n] C1 exp1 C0 exp0  n  add ... for example)
: LOAD-CONSTANTS DUP 1+ TO NEXTADD    ( store add of 2nd byte  )
  >R DUP R>                         ( DUP n under address      )
  C@ ?PAIRS          ( fetch allotted n, QUIT if not same      )
  0 DO                              ( DO n times               )
    NEXTADD C!                  ( store exponent at next byte )
    NEXTADD 1+ TO NEXTADD       ( point to next byte          )
    NEXTADD F!                  ( store coefficient           )
    NEXTADD 6 + TO NEXTADD      ( point past stored coeff.    )
    LOOP  ;
-->


Screen # 40
( Series Expansion--continued  )
( TERM raises z to the power stated in the first byte of each )
( term/constant location, starting with the value z^n of the  )
( previous term.                                               )
( z  z^n ... z  z^[n+x] z^[n+x] ,   NEXTADD points to exp.     )
: TERM
   NEXTADD C@ NEXTEXP !          ( store target exponent        )
    BEGIN
     POWER @  NEXTEXP @  = 0=   ( current exp. = target exp?   )
    WHILE                        ( if not,                      )
     FOVER F* 1 POWER +!         ( raise exponent 1, inc. POWER )
    REPEAT                       ( repeat until current=target, )
  FDUP  ;                        ( then FDUP and exit.          )
-->
OK
```

```
Screen # 41
( Series Expansion -- ADD-TO   )
( ADD-TO fetches the next coefficient and multiplies it with )
( z^n on TOS, then adds result to FACCUM, and points         )
( NEXTADD to exponent of next term.                          )
( z  z^n  z^n .... z  z^n  )
: ADD-TO
  NEXTADD 1+ TO NEXTADD        ( point to coefficent           )
  NEXTADD F@ F*                ( fetch and multiply x z^n      )
  FACCUM F+ TO FACCUM          ( add to cumulative total       )
  06 NEXTADD + TO NEXTADD ;    ( point to exp. of next term    )
-->
```

```
Screen # 42
( Series Expansion-- SERIES     )
( Used as  z [name] SERIES ... for value z.            )
( number of terms is defined in [name]                 )
: SERIES
  OVARS  >R F1 R>      ( set variables to 0, F1 to stack )
  NGET 0 DO              ( get # of terms for loop       )
          TERM  ADD-TO   ( calculate term and add to total)
          LOOP
  FDROP FDROP            ( drop calculation values       )
  FACCUM  ;              ( leave accumulated value on stack)
  ;S
```

```
Screen # 43
( SERIES expansion-- BY-N and BY-EXP  , fetching/storing exp.  )
( BY-N leaves the address of the nth exponent-coefficient pair )
( used as ....  n  NAME-CF  BY-N   followed by F@ or F!        )
: BY-N  2DUP @ >                          ( test for n inside array )
        IF CR ." n is outside array." QUIT ENDIF
        2 + SWAP 7 * + ;            ( point to coeff. in nth term)
( .. n NAME-CF BY-EXP leaves the address of coeff. with exp. n )
: BY-EXP NGET SWAP 0 FLAG !          ( n to TOS, add+1 to NEXTADD)
        BEGIN DUP NEXTADD C@ =          ( compare to term exp   )
        IF 1 ELSE OVER       FLAG @ =  ( no more than n terms )
        IF CR ." Exponent not found." QUIT ENDIF
        0 1 FLAG +! NEXTADD 7 + TO NEXTADD ENDIF ( next term  )
        UNTIL 2DROP NEXTADD 1+ ;        ;S
OK
```

```
Screen # 45
( Derivative Calculations--N derivatives of n-order polynomial.)
( Method from Hoffman, DDJ April 81, implm by W.C.Gates  Dec 82)
  F0 FVARIABLE EVAL-POINT  0 VARIABLE POLY-ARRAY
: GEN.ARRAY
```

(Continued on next page)

exponent-coefficient pairs are stored into the space. Only one header is required for all the exponents and coefficients.

Computation is done by SERIES, which expects a floating-point value and the beginning address of the data space, left by NAME-CF of the series. The first byte is the number of terms, which is used as the DO-LOOP index. TERM raises the power of the base value until it matches the next stored exponent. ADD-TO fetches the matching coefficient, multiplies it with the raised base value, and accumulates the total in ACCUM. This sequence is repeated *n* times to generate and add up *n* terms. Exponents may increase in any monotonic pattern. A series in half-power exponents, frequently encountered in flow measurement, may be generated by taking the square root of the base value before using it to compute the series.

In some applications, adaptive control for example, it may be necessary to modify the series coefficients during operation. This feature is provided by BY-N, used as "n NAME-CF BY-N," which leaves the address of the *n*th coefficient in the series. The word BY-EXP similarly leaves the address of the coefficient of the term whose exponent is *n*.

Where evaluation of polynomial derivatives is required, a slower and less compact method is implemented, in Screens 45-49 (pages 53-55). The polynomial coefficients are transferred to a working array, with floating-point 0s inserted for the absent coefficients. This array is then processed by CALC.DERIVS, resulting in the coefficient values being replaced by the *n* derivative values, including the value of the polynomial itself in the first location. The values of the *n* derivatives are then fetched as required using N DERIV. For an explanation on the mathematical method, see the article by R.A. Hoffman in *Dr. Dobb's Journal,* April 1981.

The screens provided are written for a floating-point package which uses a 6-byte format for floating-point numbers. For 4-byte formats, change the 6s and 7s to 4 and 5. The floating-point coefficients are handled as groups of bytes, so the program is indifferent to the internal sign-exponent-mantissa format.

DDJ

(Listing begins on page 52)

Series Expansion

(Listing continued, text begins on page 52)

```
  10 DUP                            ( highest anticipated order       )
  HERE DUP POLY-ARRAY !              ( beg. address to POLY-ARRAY    )
  C!                                ( store highest order in 1st byte )
  1+ 7 * 1+ ALLOT ;                 ( allot space for order+1 terms,
                          ( each of which is exp-byte plus float# )

: GET.ORDER       ( expects add. of coef. array, fetches order,  )
  DUP C@                ( which is exponent of last term, to TOS   )
  1- 7 * 1+ +
  C@ ;
GEN.ARRAY                       ( allots array when LOADed.         )
-->


Screen # 46
( Derivative Calculations -- FILL.ARRAY transfers coeff's from )
( permanent array to working array, filling F0 for absent coefs)
: FILL.ARRAY              ( poly-array-add  coef-array-add ....  )
  DUP GET.ORDER           (                    padd  cadd  n  .... )
  >R OVER R> DUP ROT C!              ( store order in 1st byte   )
  >R 1+ SWAP 1+ R> 1+ 0      ( point to exp.bytes, loop n+1 times)
  DO OVER C@ I =             ( print exp, test for equal exp's   )
   IF   DUP I SWAP C!                          ( if =, move exp. )
    1+ DUP 6 + SWAP ROT 1+ DUP 6 +        ( and calc. next add's )
    ROT ROT  SWAP >R
    F@ R> F!  SWAP
    ELSE DUP 7 + SWAP                            ( if not = stuff F0)
    DUP I SWAP C!                         ( point to next work.add)
    1+ >R F0 R> F!           ( but try again at same loc in source)
    THEN                     ( array.                              )
   LOOP 2DROP ;   -->


Screen # 47
( Derivative Calculation -- *FACS multiplies terms by n!     )

: *FACS                            ( empty stack .... same )
  F1 POLY-ARRAY @ DUP 16 + SWAP ( F1>stack, point to ^2 term)
  C@ 1+ 2                          ( n+1, 2 = for 2 to n        )
   DO
    I SWAP DUP DUP >R >R >R        ( I>stk, add's to ret. stk  )
    S->D D>F                       ( convert I to floating #    )
    F* FDUP R> F@ F*               ( x prev. "!", x coeff,      )
    R> F!                          ( store product back         )
    R> 7 +                         ( point to next coeff.       )
    LOOP
  DROP FDROP ;                     ( leave stack clean          )

-->


OK


Screen # 48
( Derivatives Calculation -- CALC.DERIVS performs calculations )
( replacing coef's with values in working array.                )
: CALC.DERIVS          ( <point of evaluation--F#> <series-name>..)
  >R EVAL-POINT F!               ( store X1 value to EVAL.POINT  )
  POLY-ARRAY @ DUP R>  ( work-add to stack, R> series-array-add)
  FILL.ARRAY C@ DUP      (   coefs into work-array, filling F0's)
  1- SWAP 0                  ( ...N-1 N  0 )
   DO DUP DUP I 1- SWAP
```

```
   DO                                ( limit is 1st index, index is  )
    I 1+ POLY-ARRAY @ BY-N F@     (  N-1 minus iteration #      )
    EVAL-POINT F@ F*                ( prev.coeff. x point value )
    I POLY-ARRAY @ BY-N DUP >R      ( add above to current coeff)
    F@ F+ R> F!
    1- -1 +LOOP DROP
   LOOP
   *FACS ;     -->                    ( each coef x n factorial)


 Screen # 49
( Derivatives Calculation-- DERIV fetches nth derivative value )
( checks for out-of bounds request, non-destructively )
: DERIV
  DUP POLY-ARRAY @ DUP
  ROT SWAP C@ >
   IF 5 SPACES ." Requested too high a derivative.   " 2DROP
   ELSE 2+ SWAP 7 * + F@
   THEN ;    ;S

           USE OF DERIVATIVES CALCULATION
 Define largest order polynomial to be used; add 1 to order and
 insert this value into Screen 45, line 4. Then 45 LOAD.

 <F#evaluation point> <series-coeff-name> CALC.DERIVS... does
 the calculations,  N  DERIV puts the nth derivative value on
 the stack.

OK


 Screen # 227
( Floating Point-- SINE  )
6 SERIES-CONSTANTS SINE-CF               ( allot storage space )
        ( coefficients                 exponents  )

-2.50521084 X -8                  11       ( monotonically  )
 2.75573192 X -6                   9       ( decreasing     )
-1.98412698 X -4                   7       ( exponents      )
 8.33333333 X -3                   5
-1.66666667 X -1                   3
 F1                                1
6 SINE-CF LOAD-CONSTANTS           ( n  [name] LOAD-CONSTANTS )
: [SINE] SINE-CF SERIES ;

( "X" is used for floating point entry to permit using "E" in)
( reading hex keypads. F1 is 1.0 floating point.            )
-->


 Screen # 228
( Trig Package -- SINE continued )

: ANGLE-PREP                       ( convert to rads if rqd       )
  FPI F/MOD FABS FIX D->S          ( get # of half circles        )
  DUP 0= IF DROP ELSE >R           ( x -1, is - for 2nd half      )
   F1        R> 0 DO               ( +1 for 1st, -1 for 2nd       )
   F1 FMINUS F* LOOP               ( apply sign to angle <180 d. )
   F* ENDIF ;                      ( calculate sine series        )

: FSIN
  DEG-RAD 0= IF DEG>RAD ENDIF      ( using degrees? convert >rads)
  ANGLE-PREP [SINE] ;              ( strip full turns, calc. and )
                                   ( apply sign, calc. value      )
;S
```

End Listing

Fast Matrix Operations
In Forth, Part 1

by Steven A. Ruzinsky

Steven A. Ruzinsky, 2110 S. Austin Blvd., Cicero, Illinois 60650.

It is fairly well-known that among various languages having software-implemented floating point, there is little practical difference in the speed at which number-intensive programs run. The reason for this small difference is that a large proportion of total program run time is spent on the floating-point routines that are written in assembly language and are therefore essentially the same for every language. Since most languages developed under this condition, it is doubtful that any need for faster numerical operations influenced their evolution. With the advent of hardware-implemented floating point in the form of numeric coprocessors such as the 8087, 80287, and 16081, a situation arises in which most of the popular numerical languages are incapable of efficiently utilizing their speed. This deficiency is most true of interpreted languages such as BASIC and APL.

In my quest for a faster alternative, I have found what I believe to be an ideal in an unpopular floating-point language, Forth. More specifically, I have found Polyforth for the IBM PC to be best for floating-point applications because it uses the 8087 register stack as an extra Forth stack and it includes the 8087 instruction set in its assembler. By using the assembler, it is not unusual for a numerically intensive program to run 100 times faster in Forth than in standard BASIC.

From the viewpoint of programming convenience, APL is the Rolls Royce of numeric languages. The enthusiasm of APL users easily matches that of Forth users, although each group loves their language for entirely different reasons. APL is characterized by syntactically simple commands that operate on entire arrays rather than on single numbers. Generally, the shortest program for performing a given task can be written in APL. APL does, however, have a reputation for being slow.

I have found that it is not at all difficult to write Forth programs that resemble APL commands. For example, consider the following phrase for performing matrix multiplication in Forth:

```
' A ' B ' C A*
```

Here, A* is the matrix multiplication operator and A, B, and C are arbitrary conformable matrices. In algebraic notation, C = AB. The advantage of Forth over APL is, of course, speed. The program represented above will multiply two 10th-order square matrices in 0.1 seconds. APL users are invited to present benchmarks for comparison.

In this first of a three-part series, some fundamental Forth matrix utilities will be presented. In the second installment, programs for the classical matrix operations such as matrix addition, multiplication, and inversion, plus some equally useful but more unusual operations, will be presented. The third part will be concerned with matrix applications. If reader response is favorable, more items on matrix applications could be forthcoming.

These programs require: (1) the IBM PC with more than 64 KB, preferably 320 KB, of RAM; (2) an 8087 numeric coprocessor installed in the empty socket next to the 8088 (dipswitch SW1-2 must be off); and (3) Polyforth Level 2 for the IBM PC from Forth, Inc.

The Programs

The fundamental programs are presented in screens 207 to 209 (page 57). A description of these programs follows:

a!, a@, etc. – Fast concatenated 16-bit variable stores and fetches. These will be used within many of the succeeding programs.

matrix – Defines a character string as a matrix. Usage example:

```
10 20 matrix X
```

Here, X is defined as a matrix of 10 rows by 20 columns. Such a matrix has the following properties:

- The data contained in the matrix resides outside the Forth dictionary and can occupy all available memory beyond the first 64 KB segment of RAM.
- Two 64-bit numbers can be stored in each matrix element. Thus, X in the above example can be considered as two interlaced matrices of real numbers or a single matrix of complex numbers. Operators subscripted with 1 and 2 will respectively apply to the first or second of the two "interlaced" matrices, whereas unsubscripted operators will apply to the matrix as a whole.
- Indexing of the matrix elements is from (0,0) to (m–1,n–1) where m is the number of rows and n is the number of columns.

variable – Defines a character string as a 16-byte variable outside the first 64 KB of RAM. Usage example:

```
variable Y
```

forget – Erases a matrix or variable in a manner analogous to the standard Forth word FORGET. Usage example:

```
forget X
```

dim@ – Retrieves the dimensions of the matrix and puts them on the parameter stack with the number of columns on top and the number of rows underneath. Usage example:

```
' X dim@
```

dim! – Changes the dimension of a matrix. The number of matrix elements, however, must not increase. Usage example:

```
200 1 ' X dim!
```

!1, !2, !! – Are matrix element and variable stores. !1 and !2 transfer the top number of the numeric stack to the indicated position in the matrix element or variable. !! is functionally equivalent to !2 !1. Usage examples:

```
3.14 5 15 X !1
2.72 1.23 Y !!
```

@1, @2, @@ – Are matrix element and variable fetches. Usage examples:

```
5 15 X @2
Y @1
```

EXC1, EXC – Are exchange matrix elements or variables. Usage examples:

```
5 15 X   3 2 X   EXC1
Y   8 9 X   EXC1
```

A few final notes: Because floating-point numbers and integers are stored on separate stacks, their order before an operator makes no difference. Thus, these phrases are equivalent:

```
123.456  654.321  3  7  X!!
3  7  123.456  654.321  X!!
3  123.456  7  654.321  X!!
```

Also, the use of ' should be commented on. This is called "tick" and it places the dictionary address of the word following it

onto the parameter stack. ' is for use outside a colon definition. Inside a colon definition ['] must be substituted. ' and ['] are used extensively to pass subroutines.

Mr. Ruzinsky, in an attempt to demonstrate the floating-point speed that one can achieve with Forth, brought to our attention the article "Benchmarking the 8087 Numeric Coprocessor" in the March 1983 issue of Personal Computer Age. *In it, G. Scott Owens provided some benchmarks for the IBM PC with an 8087 coprocessor (and some for the IBM and other machines without the 8087). The fastest times obtained for his floating-point routine (a mixture of various functions such as sine, cosine, and squareroot) were: IBM Pascal with an 8087 – 6 seconds; and compiled IBM BASIC with an 8087 – 10 seconds for single precision and 13 seconds for double precision. (The 8087 software in those cases was from Microware.) Mr. Ruzinsky wrote a Forth version on his own system of the floating-point routine used by Mr. Owens and obtained a time of 2.625 seconds. His Forth routine is printed in Figure 1 (above).*

```
316
( FPBENCH in Polyforth, Double Prec., 2.625 sec./500 iters.    )
( Note: SIN, COS, EX, #IN, and 4ARRAY are author written       )
( routines that are not included in standard Polyforth.        )

: 4ARRARY CREATE 8 * 10 + ALLOT ;CODE W INC W INC 0 POP 0 SHL
          0 SHL 0 SHL W 0 ADD 0 PUSH NEXT
            10  4ARRAY  X       LVARIABLE A
: LOOP2  10 1 DO I X L@ FDUP F* PI F/ A L! LOOP ;
: SBR    PI 2.1 F/ FDUP SIN 1 X L! FDUP COS
         2 X L!  FDUP FDUP F* 3 X L!  FDUP FSQRT 4 X L!
         FDUP EX 5 X L! FDUP LN 6 X L!   FDUP 7 X L!
         FDUP PI F* 8 X L!  SIN 2.0 F* 9 X L! ;
: FPBENCH  CR ." THIS IS A NUMBER CRUNCHING SPEED TEST "
           CR ." ENTER THE NUMBER OF ITERATIONS " #IN
           COUNTER SWAP 0 DO SBR LOOP2 LOOP COUNTER SWAP
           - >N 1000.0 F/ CR ." ELAPSED TIME = " 3 F. ;
```

Figure 1.

Floating-point benchmark routine in Forth.

DDJ

(Listing begins below)

Fast Matrix Operations

(Text begins on page 56)

```
207 LIST

 VARIABLE a           CODE a! 0 POP a  STA NEXT
                      CODE a@ a LDA 0 PUSH NEXT
 VARIABLE b           CODE b! 0 POP b  STA NEXT
                      CODE b@ b LDA 0 PUSH NEXT
 VARIABLE c           CODE c! 0 POP c  STA NEXT
                      CODE c@ c LDA 0 PUSH NEXT
 VARIABLE i           CODE i! 0 POP i  STA NEXT
                      CODE i@ i LDA 0 PUSH NEXT
 VARIABLE j           CODE j! 0 POP j  STA NEXT
                      CODE j@ j LDA 0 PUSH NEXT
 VARIABLE k           CODE k! 0 POP k  STA NEXT
                      CODE k@ k LDA 0 PUSH NEXT
 VARIABLE m           CODE m! 0 POP m  STA NEXT
                      CODE m@ m LDA 0 PUSH NEXT
 VARIABLE n           CODE n! 0 POP n  STA NEXT
                      CODE n@ n LDA 0 PUSH NEXT

208 LIST

 ( Matrix Defining Operators )
 ~ VARIABLE (here) 1 (here) !
 ~ : here (here) @ ;
 ~ : forget -' ABORT" ?"  B/H - DUP 'S H 2+ @ WITHIN
            CAN'T H ! CURRENT CONTEXT 2+ DO I BEGIN
            @ DUP HERE < UNTIL I ! 2 /LOOP @ (here) ! ;
 ~ : allot here + (here) ! ;
 ~ : variable here CONSTANT 1 allot ;
 ~ : matrix CREATE here , SWAP 2DUP , , * allot ;CODE 0 POP
            1 POP 2 W) 1 ADD 4 W) MUL 1 0 ADD 0 PUSH NEXT
 VARIABLE adr            ~ CODE adr@ adr LDA 0 PUSH NEXT
```

(Continued on next page)

Fast Matrix Operations (Listing continued, text begins on page 56)

```
                             ~ CODE adr! 0 POP adr STA NEXT
 ~ CODE dim@ W POP W ) 0 MOV adr STA 2 W) 0 MOV 0 PUSH
             4 W) 0 MOV 0 PUSH NEXT
 ~ CODE dim! W POP W ) 0 MOV adr STA 0 POP 0 4 W) MOV 0 POP
             0 2 W) MOV NEXT

209 LIST

 ( Matrix Element Fetch, Store and Exchange )
 CODE !1 0 DS SSG DS POPS R64 65520 FSTP 0 DS LSG NEXT
 CODE @1 0 DS SSG DS POPS R64 65520 FLD 0 DS LSG NEXT
 CODE !2 0 DS SSG DS POPS R64 65528 FSTP 0 DS LSG NEXT
 CODE @2 0 DS SSG DS POPS R64 65528 FLD 0 DS LSG NEXT
 CODE !! 0 DS SSG DS POPS R64 65528 FSTP R64 65520 FSTP
         0 DS LSG NEXT
 CODE @@ 0 DS SSG DS POPS R64 65520 FLD R64 65528 FLD
         0 DS LSG NEXT
 ~ CODE EXC 65520 # W MOV 0 DS SSG DS POPS R64 8 W) FLD R64
   W ) FLD 2 DS SSG DS POPS R64 8 W) FLD R64 W ) FLD 1 DS SSG
   2 DS LSG R64 W ) FSTP R64 8 W) FSTP 1 DS LSG R64 W ) FSTP
   R64 8 W) FSTP 0 DS LSG NEXT
 CODE EXC1 65520 # W MOV 0 DS SSG DS POPS R64 W ) FLD 2 DS SSG
   DS POPS R64 W ) FLD 1 DS SSG 2 DS LSG R64 W ) FSTP 1 DS LSG
   R64 W ) FSTP 0 DS LSG NEXT
```

End Listing

Yes, You Can

Trace Through BDOS

Letters to Dr. Dobb have shown an interest in using the DDT Trace function to examine the operation of BDOS functional calls. The last word on the subject (February 1983, page 11) was, "This will never work no matter how clever one is. . . ." Who could resist a challenge like that?

Demonstrating the Problem

Let's try to Trace through a very simple BDOS function and see what the problem is. BDOS function 31 is a good candidate for this because it only fetches the disk parameter address and puts it in the H/L register pair. The test program (TRACE.ASM – shown in Listing 1 below) contains the minimum elements necessary to demonstrate the problem.

(1) It initially clears the H/L pair (so we know the non-zero result was not left over from a previous run).

(2) It calls BDOS function 31.

(3) It executes an instruction (NOP) after it returns.

(4) It has an easily recognized end.

The DDT session is separated into procedures to make it easier to follow.

In Procedure 1 (page 61), I booted a 48K system (system size is important) and used DDT to get the test program in memory. The List command disassembled the object code in memory, showing that the program was actually there. The Go command executed the program (from 100 to 108) and the Examine (X) command showed that the H/L pair had B47E in it when the test program was finished. B47E is in fact the address of the disk parameter header for this particular CP/M configuration. If you run this program on your system, you will probably get a different number in the H/L pair. If you are skeptical, you can examine memory beginning at that location and verify that it is actually a disk parameter header.

The G100,100 command gets the program counter back to the beginning of the program, and T0C is the command to Trace twelve lines (DDT thinks in hex). Notice that the H/L pair was cleared, Register C was set to 31 (1F hex), and the BDOS routine was called.

by Do-While Jones

Do-While Jones, 324 Traci Lane, Ridgecrest, California 93555.

Location 0005 always contains a jump to the BDOS section of CP/M. Since the location of BDOS depends upon memory size, this address will vary depending upon system size. Whenever you boot the CP/M system, it will always make sure the jump command at 0005 is set to the proper address. In the 48K system I was using, this address was 9500. The Trace display showed the jump to 9500, and the first few instructions in BDOS, but the line following JMP A506 shows a NOP. Closer inspection shows the program counter appeared to go from 9BA7 directly to 0108. I say "appeared" because the intermediate steps really were executed. We can tell this by the fact that the H/L pair has the right answer (B47E) in it.

The invisible gap between 9BA7 and 0108 is the sequence of instructions that some readers are interested in seeing.

To Trace or Not To Trace . . .

Although it is possible to Trace through BDOS, I do not recommend that you do it routinely. DDT is designed not to Trace through BDOS functions *for your convenience*. If you defeat this feature frequently, you will just make your own life more difficult.

Inexperienced programmers use Trace at the first sign of trouble. They hit control-P, T100 and start looking through reams of paper that slowly emerge from the printer. That is not a very efficient way to debug a program, especially if it contains something equivalent to "FOR I = 1 TO 500."

The easier way to debug a program is to use break points. Use the "G" command to execute a small portion of the program. When DDT finishes executing that section, it will return with the "*" prompt, which is your cue to use "X" to examine registers or "D" to display memory. If you have written your program in modular form, there will be longer places to stop and examine the results. For example, if you have a module which is supposed to convert hex numbers into BCD form, then you can stop the program just before the routine to see what hex number is going into the routine and see if the BCD number that came out is

Simple Program for Trace Demonstration

```
                ;               TRACE.ASM
                ;               16 FEBRUARY 1983
                ;               DO-WHILE JONES

                ;       THIS IS USED TO DEMONSTRATE HOW TO TRACE THROUGH
                ;       BDOS CALLS.

                ;       GLOBAL ADDRESSES

0005 =                  BDOS    EQU     0005H

0100                    ORG 100H

0100 210000             LXI H,0         ; CLEAR THE H/L PAIR
0103 0E1F               MVI C,31        ; GET ADDRESS OF DISK PARAMETERS
0105 CD0500             CALL BDOS       ; USING BDOS COMMAND 31
0108 00                 NOP             ; DUMMY OP CODE
0109 C30901     SELF:   JMP SELF        ; DO FOREVER LOOP

010C                    END
```

Listing 1.

correct. Don't Trace through the whole routine *unless* the BCD answer is wrong. Then it makes sense to Trace to see why it gave the wrong answer. If you try to use "G" to execute a portion of a routine and never see the " * " prompt, it means the program has gone to "never-never land." Then you might want to Trace to see where you PUSHed without a POP in a subroutine, or where you selected the wrong condition for a conditional jump. But it is best to try to isolate the error using break points before using Trace.

Do-While's Rule:
Never Trace more than you have to.

DDT Saves You From Yourself

The good folks at Digital Research put a lot of thought into the DDT program. They foresaw users trying to Trace through a program which contains a BDOS call to output to the console (or worse yet, read a sector). They knew how frustrated the user would be if the screen filled up with dozens of lines of BDOS code. On those rare occasions when I have resorted to Trace, and have happened to Trace through a BDOS command, I have been very grateful that DDT turned off the display until the BDOS function returned control to my own program.

Normally you should not care what the BDOS does internally. All you care about is that the return parameters are correct. You can find that out by putting a break point immediately after the BDOS call. Then you can examine registers (or memory) to see what it did, without having to try to figure out how it did it. Be sure to check the flag register. There are times when BDOS returns with the accumulator = 00, but the zero flag is *not* set. Since the JZ (Jump on Zero) instruction looks at the zero flag rather than the contents of the accumulator, your program may not branch to the desired routine unless you checked the zero flag with ANA A.

Is There Ever A Proper Time to Trace BDOS Calls?

There has only been one time that I ever found it necessary to Trace a BDOS function. I wrote an extension to CP/M 2.2 which allowed data to be displayed simultaneously on the console screen and stored in a disk file. In this case I tried to call some BDOS routines from BIOS, and the program crashed because the BDOS is not reentrant. I needed to find out why it was crashing and how to avoid it. I Traced through enough of BDOS to find out what prevented the subroutine from returning, and fortunately I was able to find other entry points which allowed me to use the BDOS subroutines to write a sector and close a file. Procedure 2 (page 62) shows the method I used.

The Trick – Two CP/M Systems at Once

After demonstrating the problem using Procedure 1, I went back to the sys-

Procedure 1.

```
48K VERSION 2.2 CP/M

A>DDT TRACE.HEX
DDT VERS 2.2
NEXT  PC
010C 0000
-L100,10B
  0100  LXI  H,0000
  0103  MVI  C,1F
  0105  CALL 0005
  0108  NOP
  0109  JMP  0109
  010C
-G100,109
*0109
-X
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0109 JMP  0109
-
-G100,100
*0100
-T0C
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0100 LXI  H,0000
C0Z1M0E1I0 A=7E B=B400 D=0000 H=0000 S=0100 P=0103 MVI  C,1F
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0000 S=0100 P=0105 CALL 0005
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0000 S=00FE P=0005 JMP  9500
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0000 S=00FE P=9500 JMP  9BA2
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0000 S=00FE P=9BA2 XTHL
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0108 S=00FE P=9BA3 SHLD A44A
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0108 S=00FE P=9BA6 XTHL
C0Z1M0E1I0 A=7E B=B41F D=0000 H=0000 S=00FE P=9BA7 JMP  A506
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0108 NOP
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0109 JMP  0109
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0109 JMP  0109*0109
-G0
```

tem and PIPed the TRACE.HEX file over to another disk, which had a different size CP/M system on it. I had an old disk with a 32K system on it that I had SYSGENed when I only had 32K of memory in my system. So, *without turning off the power,* I cold-booted this 32K disk. This put a 32K CP/M system in memory, while leaving a copy of the 48K system there, too.

Procedure 2 gave results that were similar to Procedure 1, but they were not identical. The H/L pair returned the value 747E, which is exactly 16K lower than B47E because the 32K system is 16K smaller than the 48K system. Also, the JMP instruction at location 0005 sent the program to 5500 instead of 9500.

The BDOS routine at 9500 is still in memory but DDT doesn't know that. So look what happens if we change CALL 0005 to CALL 9500. We can reset the program counter to the beginning of the test program with G100,100 and Trace a bunch of lines (36 hex is enough).

This time it's all there! After the JMP A506, the program counter goes to A506, rather than skipping down to 0108.

The group of instructions up to P= A546 PCHL is the BDOS entry routine which stores parameters, sets up the BDOS stack, and jumps to the selected BDOS routine. BDOS function 31 itself is only three lines.

```
LHLD B2BB
SHLD A845
RET
```

The nine lines after that are the BDOS exit routine, which restores parameters, restores the user stack, and fetches the answer from A845.

The desired Trace is shown in Procedure 3 on page 64.

Why It Works

Using a second CP/M system does two things that permit a successful Trace of BDOS functions:

(1) DDT uses the 32K BDOS stack and 32K BDOS reserved memory locations while the test program is using the 48K BDOS stack and reserved memory locations.

(2) DDT inhibits the display when the user program counter is in the 32K BDOS, but it doesn't care that the program counter is in the reserved area of the 48K BDOS. Therefore it permits the program flow to be displayed.

There's nothing really magical about 48K and 32K. You could use any two CP/M systems, providing they don't overlap. You probably have a 20K (or 24K)

Procedure 2.

```
A)PIP B:=TRACE.HEX

32K VERSION 2.2 CP/M


A)DDT TRACE.HEX
DDT VERS 2.2
NEXT  PC
010C 0000
-L100,10B
  0100  LXI   H,0000
  0103  MVI   C,1F
  0105  CALL  0005
  0108  NOP
  0109  JMP   0109
  010C
-G100,109
*0109
-X
C0Z1M0E1I0 A=7E B=7400 D=0000 H=747E S=0100 P=0109 JMP  0109
-
-G100,100
*0100
-T0C
C0Z1M0E1I0 A=7E B=7400 D=0000 H=747E S=0100 P=0100 LXI  H,0000
C0Z1M0E1I0 A=7E B=7400 D=0000 H=0000 S=0100 P=0103 MVI  C,1F
C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=0100 P=0105 CALL 0005
C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=0005 JMP  5500
C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=5500 JMP  5BA2
C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=5BA2 XTHL
C0Z1M0E1I0 A=7E B=741F D=0000 H=0108 S=00FE P=5BA3 SHLD 644A
C0Z1M0E1I0 A=7E B=741F D=0000 H=0108 S=00FE P=5BA6 XTHL
C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=5BA7 JMP  6506
C0Z1M0E1I0 A=7E B=7400 D=0000 H=747E S=0100 P=0108 NOP
C0Z1M0E1I0 A=7E B=7400 D=0000 H=747E S=0100 P=0109 JMP  0109
C0Z1M0E1I0 A=7E B=7400 D=0000 H=747E S=0100 P=0109 JMP  0109*0109
```

I WILL BEAT ANY COMPETITOR'S PRICE PROVIDED IT IS NOT BELOW MY COST. **TRY** TO BEAT THESE IC PRICES:

DYNAMIC RAM		
64K	200 ns	$4.99
64K	150 ns	5.09
64K	120 ns	5.90
16K	200 ns	1.35
EPROM		
2764	300 ns	$7.25
2732	450 ns	4.15
2716	450 ns	3.20
2532	450 ns	4.60
STATIC RAM		
6116P-3	150 ns	$4.10
2016-LIKE	150 ns	4.40
2114	200 ns	1.45
Z80A FAMILY		
CPU, CTC, or PIO		$3.39
DART		8.25
DMA or SIO/0		12.50

MasterCard/VISA or UPS CASH COD

Factory New, Prime Parts

Prices 5/1/83

MICROPROCESSORS UNLIMITED

24,000 South Peoria Ave.
BEGGS, OK. 74421

(918) 267-4961

Prices subject to change. Call for volume prices. Subiect to available quantities. Shipping & Insurance extra. Cash discount prices shown.

Circle **no. 53** on reader service card.

Uniforth

One of the finest implementations of the FORTH language. Field tested and reliable, **UNIFORTH** is available for the **IBM PC** as well as most systems with 8" disks and the following processors:

8080	PDP-11
Z80	68000
8086/8	16032

As a task, **UNIFORTH** is compatible with and supports all features and file types of the CP/M, CDOS, MS-DOS and DEC operating systems. *As an operating system,* **UNIFORTH** will function "stand-alone" on most commercial microcomputers.

The FORTH-79 Standard language has been extended with over 500 new words that provide full-screen and line-oriented editors, array and string handling, enhanced disk and terminal I/O, and an excellent assembler. Detailed reference manuals supply complete documentation for programming and system operation, in an easy-to-understand, conversational style using numerous examples.

Optional features include an excellent floating-point package with all transcendental functions (logs, tangents, etc.), the MetaFORTH cross-compiler, printer plotting and CP/M file transfer utilities, astronomical and amateur radio applications, word processing, etc.

Compare these features with any other FORTH on the market:

- Speed and efficiency
- Variety of options
- Ease of use
- Documentation quality

You'll find **UNIFORTH** is superior.

Prices start at $35. Call or write for our free brochure.

Unified Software Systems

P.O. Box 2644, New Carrollton, MD 20784, (301) 552-1295

Circle **no. 85** on reader service card.

We Deliver.

At Key Micro Systems, Inc. we deliver simply the best S-100 computers for expandable multi-user or single user systems. You can run any mix of 8 or 16-bit software on any terminal. And our use of the leading, highest quality peripherals combined with the power of CompuPro boards assures you of complete system confidence.

Whether you need an established system, peripheral support, or a system configured to your particular specs, we deliver. On time. Every time.

KEY MICRO SYSTEMS INC.

Your Authorized CompuPro SYSTEMS CENTER

1606 Nooseneck Hill Rd., PO Box 715, Coventry, RI 02816 · 401/828-7270
822 Boylston St., Suite 201, Chestnut Hill, MA 02167 · 617/738-7305

Circle **no. 43** on reader service card.

A Professional Quality Z80/8080 Disassembler

REVAS Version 3

Uses either ZILOG or 8080 mnemonics
Includes UNDOCUMENTED Z80 opcodes
Handles both BYTE (DB) & WORD (DW) data
Disassembles object code up to 64k long!
Lets you insert COMMENTS in the disassembly!

A powerful command set gives you:

INTERACTIVE disassembly
Command Strings & Macros
On-line Help
Calculations in ANY Number Base!
Flexible file and I/O control
All the functions of REVAS V2.5

REVAS:

Is fully supported with low cost user updates
Runs in a Z80 CPU under CP/M*
Is normally supplied on SSSD 8" diskette

Revas V 3...$90.00 Manual only...$15.00

California Residents add 6½% sales tax

REVASCO
6032 Charlton Ave., Los Angeles, CA. 90056
(213) 649-3575

*CP/M is a Trademark of Digital Resaerch, Inc.

Circle **no. 71** on reader service card.

"distribution system" that you could use in conjunction with your full-size system.

It doesn't matter which you load first if there is sufficient space between the systems. But if the systems are nearly of equal size (like 24K and 32K), you might have trouble if you load the smaller one first. DDT moves itself to a position just below the CP/M system, so if it tries to put itself just below a 32K system, it might land on top of the 24K system. If you boot the bigger system first (as I did in the example) and then DDT from the smaller system, DDT will be below the smaller system, so it can't interfere with the larger system.

They Said It Couldn't Be Done

The previous letters to Dr. Dobb said that it is impossible to Trace in the BDOS because it is not reentrant, so "attempts to trace through BDOS will result in the saved values being overwritten and destroyed by the recursive BDOS calls made by the debugger. This tends to send you to a tight loop in never-ever land." This is true. But just because something is impossible doesn't mean that it can't be done. Every good stage magician knows that. If you can create the illusion that it is being done, that is good enough.

DDJ

Procedure 3.

```
-S106
0106 05 00
0107 00 95
0108 00 .
-L100,10B
  0100  LXI   H,0000
  0103  MVI   C,1F
  0105  CALL  9500
  0108  NOP
  0109  JMP   0109
  010C
 -G100,100
 *0100
 -T36
 C0Z1M0E1I0 A=7E B=7400 D=0000 H=747E S=0100 P=0100 LXI  H,0000
 C0Z1M0E1I0 A=7E B=7400 D=0000 H=0000 S=0100 P=0103 MVI  C,1F
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=0100 P=0105 CALL 9500
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=9500 JMP  9BA2
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=9BA2 XTHL
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0108 S=00FE P=9BA3 SHLD A44A
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0108 S=00FE P=9BA6 XTHL
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=9BA7 JMP  A506
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=A506 JMP  A511
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=A511 XCHG
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=A512 SHLD A843
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=A515 XCHG
 C0Z1M0E1I0 A=7E B=741F D=0000 H=0000 S=00FE P=A516 MOV  A,E
 C0Z1M0E1I0 A=00 B=741F D=0000 H=0000 S=00FE P=A517 STA  B2D6
 C0Z1M0E1I0 A=00 B=741F D=0000 H=0000 S=00FE P=A51A LXI  H,0000
 C0Z1M0E1I0 A=00 B=741F D=0000 H=0000 S=00FE P=A51D SHLD A845
 C0Z1M0E1I0 A=00 B=741F D=0000 H=0000 S=00FE P=A520 DAD  SP
 C0Z1M0E1I0 A=00 B=741F D=0000 H=00FE S=00FE P=A521 SHLD A80F
 C0Z1M0E1I0 A=00 B=741F D=0000 H=00FE S=00FE P=A524 LXI  SP,BE00
 C0Z1M0E1I0 A=00 B=741F D=0000 H=00FE S=BE00 P=A527 XRA  A
 C0Z1M0E1I0 A=00 B=741F D=0000 H=00FE S=BE00 P=A528 STA  B2E0
 C0Z1M0E1I0 A=00 B=741F D=0000 H=00FE S=BE00 P=A52B STA  B2DE
 C0Z1M0E1I0 A=00 B=741F D=0000 H=00FE S=BE00 P=A52E LXI  H,B274
 C0Z1M0E1I0 A=00 B=741F D=0000 H=B274 S=BE00 P=A531 PUSH H
 C0Z1M0E1I0 A=00 B=741F D=0000 H=B274 S=BDFE P=A532 MOV  A,C
 C0Z1M0E1I0 A=1F B=741F D=0000 H=B274 S=BDFE P=A533 CPI  29
 C1Z0M1E0I0 A=1F B=741F D=0000 H=B274 S=BDFE P=A535 RNC
 C1Z0M1E0I0 A=1F B=741F D=0000 H=B274 S=BDFE P=A536 MOV  C,E
 C1Z0M1E0I0 A=1F B=7400 D=0000 H=B274 S=BDFE P=A537 LXI  H,A547
```

(Continued on top of page 65)

```
C1Z0M1E0I0 A=1F B=7400 D=0000 H=A547 S=BDFE P=A53A MOV  E,A
C1Z0M1E0I0 A=1F B=7400 D=001F H=A547 S=BDFE P=A53B MVI  D,00
C1Z0M1E0I0 A=1F B=7400 D=001F H=A547 S=BDFE P=A53D DAD  D
C0Z0M1E0I0 A=1F B=7400 D=001F H=A566 S=BDFE P=A53E DAD  D
C0Z0M1E0I0 A=1F B=7400 D=001F H=A585 S=BDFE P=A53F MOV  E,M
C0Z0M1E0I0 A=1F B=7400 D=0026 H=A585 S=BDFE P=A540 INX  H
C0Z0M1E0I0 A=1F B=7400 D=0026 H=A586 S=BDFE P=A541 MOV  D,M
C0Z0M1E0I0 A=1F B=7400 D=B226 H=A586 S=BDFE P=A542 LHLD A843
C0Z0M1E0I0 A=1F B=7400 D=B226 H=0000 S=BDFE P=A545 XCHG
C0Z0M1E0I0 A=1F B=7400 D=0000 H=B226 S=BDFE P=A546 PCHL
C0Z0M1E0I0 A=1F B=7400 D=0000 H=B226 S=BDFE P=B226 LHLD B2BB
C0Z0M1E0I0 A=1F B=7400 D=0000 H=B47E S=BDFE P=B229 SHLD A845
C0Z0M1E0I0 A=1F B=7400 D=0000 H=B47E S=BDFE P=B22C RET
C0Z0M1E0I0 A=1F B=7400 D=0000 H=B47E S=BE00 P=B274 LDA  B2DE
C0Z0M1E0I0 A=00 B=7400 D=0000 H=B47E S=BE00 P=B277 ORA  A
C0Z1M0E1I0 A=00 B=7400 D=0000 H=B47E S=BE00 P=B278 JZ   B291
C0Z1M0E1I0 A=00 B=7400 D=0000 H=B47E S=BE00 P=B291 LHLD A80F
C0Z1M0E1I0 A=00 B=7400 D=0000 H=00FE S=BE00 P=B294 SPHL
C0Z1M0E1I0 A=00 B=7400 D=0000 H=00FE S=00FE P=B295 LHLD A845
C0Z1M0E1I0 A=00 B=7400 D=0000 H=B47E S=00FE P=B298 MOV  A,L
C0Z1M0E1I0 A=7E B=7400 D=0000 H=B47E S=00FE P=B299 MOV  B,H
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=00FE P=B29A RET
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0108 NOP
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0109 JMP  0109
C0Z1M0E1I0 A=7E B=B400 D=0000 H=B47E S=0100 P=0109 JMP  0109*0109
-G0
```

Julian Dates for Microcomputers

Julian dates have long been used by astronomers to facilitate accurate reckoning of dates over long periods of time, and they can also be extremely useful in various applications of computers. A Julian date is simply the number of a given day counted in sequence from some base date which is assigned Julian date 0. For example, if 1 January 1983 is taken as the base date 0, then 2 January 1983 would have Julian date 1, 3 January 1983 would have Julian date 2, and 1 January 1984 would have Julian date 365.

There are at least four reasons why Julian dates can be usefully applied in computer programs (inventive computerists can probably find more):

(1) The Julian date provides a compact and economical way of storing a date as opposed to representing it as a string of ASCII characters.

(2) Any two dates may be simply compared by an arithmetic test to determine which date is earlier. This is especially useful when items are to be sorted by date.

(3) The exact number of days between any two dates can be determined by a simple subtraction.

(4) When a Julian date is divided by 7, the remainder gives the corresponding day of the week (this will be explained in more detail below).

True Julian dates as used by astronomers take noon, 1 January 4713 B.C. as the base date (astronomers count days from noon to noon). But for most computer work, it's more useful to take a year someplace in the current century as a base year, because there normally is no need to reference dates very far back in the past. A base date in 1900, for example, would be sufficient for the Julian date representation of employee birth dates in almost any business today. Furthermore, restricting the forward span of dates from the base date to some reasonable future limit allows the Julian date to be stored and manipulated economically. For example, choosing to store a Julian date in a 16-bit word (two bytes) allows a span of 65,536 days, which is approximately 179.4 years. The algorithms for Julian date conversion given below, which use a 16-bit Julian date, allow a base year to be chosen between 1900 and 1920, thus giving a terminating date between 2079 and 2099 in the next century.

In order to use Julian dates on a computer, two conversion routines are needed: CTOJ, which converts a calendar date to a Julian date, and JTOC, which converts a Julian date back to a calendar date. The routines given here are based on Algorithm 199 in *The Collected Algorithms of the ACM*, as presented by R. G. Tantzen in 1963. Calendar dates are given in the form DAY (an integer from 1 to 31), MONTH (an integer from 1 to 12), and YEAR (an integer from 1900 to 2099). The Julian date is given (or returned) as the 16-bit unsigned integer JDATE, ranging from 0 (Julian date 0) to 65,535 (last day).

The base date is 1 March 1900, or 1 March of any leap year after 1900 not greater than 1920. The first of March rather than the first of January is used as a base date in order to avoid problems with 29 February in leap years. For similar reasons, the routines as given will not work for dates before 1900 or after 2099 (1900 and 2100 are not leap years, while 2000 is). Readers interested in representing dates outside these ranges should refer to Tantzen's original algorithms, which handle any Gregorian calendar date (but not within 16 bits).

The conversion routines are shown first in a pseudo higher-level language to make the algorithms clear (see Figure 1, below). But the reader should be cautioned that they cannot be directly translated into any higher-level language which, like many BASICs, doesn't support long (32-bit) integer arithmetic. Though the results are all 16-bit quantities (or less), some of the intermediate calculation requires 32-bit arithmetic, and the algorithms also depend on the properties of integer division.

As illustrated in Figure 1, the normal month number is converted to the number of a month in a pseudo year running from 1 March through 28 (or 29) February, and the year number is adjusted if necessary. The magic number 1461 which appears in CTOJ is simply the number of days in a leap-year cycle of three years of 365 days each and one year of 366 days. Therefore the expression (1461 * y)/4 gives the total number of days in all preceding years from the base year up to the specified year. The integer function (153*m + 2)/5 similarly gives the total number of days in any pseudo year up to, but not including, month m. Therefore, adding DAY to the sum of the

Conversion of Calendar Date to Julian Date

```
CTOJ:   y = YEAR - Base Year
        If MONTH .gt. 2,  m = MONTH - 3
             else  m = MONTH + 9,  y = y - 1
        JDATE = (1461*y)/4 + (153*m + 2)/5 + DAY - 1
```

Conversion of Julian Date to Calendar Date

```
JTOC:   y = (4*JDATE + 3)/1461
        d = (4*JDATE + 3) mod 1461
        YEAR = y + Base Year
        d = d/4 + 1
        m = (5*d - 3)/153
        d = (5 * d - 3) mod 153
        DAY = d/5 + 1
        If m .lt. 10,  MONTH = m + 3
             else  MONTH = m - 9,  YEAR = YEAR + 1
```

Figure 1.

by Gordon King

Gordon King, King Software, P.O. Box 208, Red Bank, New Jersey 07701.

two preceding expressions gives the day number of the specified calendar date; and finally subtracting 1 gives the base-zero Julian date. The routine JTOC simply reverses the above calculation to convert back to calendar date.

The JDATE numbers used by these algorithms may also be converted to day numbers in other similar date systems by the addition or subtraction of a suitable conversion constant. Suppose that FDATE is such a day number in some "foreign" date system which also numbers days sequentially, but from a different base date. Then

$$JDATE = FDATE + CC$$

and

$$FDATE = JDATE - CC$$

where CC is the conversion constant, which is simply the JDATE value for the base date of the foreign system; it must of course lie within the valid JDATE range. For example, MP/M and CP/M Plus use a 16-bit integer date with 1 January 1978 taken as Day 1 (hence 31 December 1977 is Day 0). The JDATE value for 31 December 1977 is 28,429 if 1900 is used as the JDATE base year, or 21,124 for base year 1920. Therefore simply adding the appropriate one of these two constants to MP/M's date at the beginning of the routine JTOC converts it to the corresponding JDATE and results in the correct calendar date. Conversely, subtracting it from JDATE at the end of the routine CTOJ gives back the MP/M-form date corresponding to the given calendar date (calendar dates before 31 December 1977 will result in negative MP/M dates, which may have some useful interpretation outside of MP/M itself).

Before going on to present the two routines in Z80 assembly language, mention should be made of the method of determining the day of the week corresponding to a given Julian date. When a Julian date is divided by 7, the remainder will be one of the seven numbers from 0 to 6 (the quotient is ignored). These correspond to the seven days of the week in order, beginning from a day which depends on the base year. For base year 1900, a remainder of 0 means Thursday, 1 means Friday, and so on down to 6, which means Wednesday. For base year 1920, a remainder of 0 means Monday, 1 means Tuesday, etc. Consequently, these remainders can be used to index into a suitably ordered table of day names when it's required to print the day of the week corresponding to a given date (after it's been converted to a Julian date, of course).

Finally, the listing (page 68) provides the two conversion subroutines in Z80 assembly language. The date is transferred in the byte DAY, the byte MONTH, and

the word (two bytes) YEAR, while the Julian date is transferred in the word JDATE. Since the Julian date is contained in a 16-bit word, the legal range of dates is from 1 March 1900 to 4 August 2079 if 1900 is chosen as the base year, or from 1 March 1920 to 4 August 2099 if 1920 is chosen as the base.

The subroutine CTOJ checks DAY, MONTH, and YEAR for legal values and jumps to a routine ERROR (not supplied) in case they're out of range. Two subroutines are also supplied for integer multiplication and division, but if date conversions are to be done heavily, as in an inner loop, some improvement in performance could be gained by using shifts and/or adds to do the multiplications by 4 and 5 and the divisions by 4. DDJ

(Listing begins below)

Julian Dates (Text begins on page 66)

```
        YEAR0   = 1900          ;BASE YEAR

;CONVERT CALENDAR DATE TO JULIAN


CTOJ:   LD      HL,(YEAR)       ;GET THE YEAR
        LD      A,(MONTH)       ;AND THE MONTH
        OR      A,A             ;CHECK FOR A LEGAL VALUE
        JP      Z,ERROR
        CP      A,12+1
        JP      NC,ERROR
        SUB     A,3             ;MONTH IN MAR-FEB "YEAR"
        JR      NC,CTOJ1
        ADD     A,12            ;JAN OR FEB BECOME 10, 11
        DEC     HL              ;   OF PREVIOUS YEAR
CTOJ1:  LD      E,A             ;PUT MONTH IN DE
        LD      D,0
        LD      BC,-YEAR0       ;SUBTRACT THE BASE YEAR
        ADD     HL,BC
        LD      A,H             ;CHECK FOR LEGAL YEAR
        OR      A,A
        JP      NZ,ERROR
        LD      A,L
        CP      A,179+1
        JP      NC,ERROR
        PUSH    DE              ;SAVE THE MONTH
        LD      DE,1461         ;DAYS IN A LEAP-YEAR CYCLE
        CALL    MULWW           ; DE*HL TO (DE,HL)
        LD      BC,4
        CALL    DIVLW           ; (DE,HL)/BC TO DE (QUOTIENT)
        POP     HL              ;GET THE MONTH BACK
        PUSH    DE              ;SAVE  (YEAR*1461)/4
        LD      DE,153
        CALL    MULWW           ; DE*HL TO (DE,HL)
        INC     HL              ;ADD 2
        INC     HL              ; (NO OVERFLOW POSSIBLE)
        LD      BC,5
        CALL    DIVLW           ; (DE,HL)/BC TO DE (QUOTIENT)
        POP     HL              ;GET BACK FIRST TERM
        ADD     HL,DE           ;ADD (153*MONTH + 2)/5
        JP      C,ERROR         ;ERROR IF OVERFLOW
        LD      A,(DAY)         ;GET DAY
        DEC     A               ;REDUCE BY ONE FOR BASE-0 J.D.
        CP      A,31            ;CHECK FOR LEGAL VALUE
```

```
          JP        NC,ERROR
          LD        E,A                 ;TO DE
          LD        D,0
          ADD       HL,DE               ;ADD IN
          JP        C,ERROR             ;ERROR IF OVERFLOW
          LD        (JDATE),HL          ;STORE AWAY
          RET



;CONVERT JULIAN DATE BACK TO CALENDAR DATE

JTOC:     LD        HL,(JDATE)          ;GET THE JULIAN DATE
          LD        DE,4                ;4*JDATE + 3
          CALL      MULWW               ; DE*HL TO (DE,HL)
          LD        BC,3
          ADD       HL,BC
          JR        NC,JTOC1
          INC       DE
JTOC1:    LD        BC,1461             ;y = (4*JDATE + 3)/1461
          CALL      DIVLW               ; (DE,HL)/BC TO DE(Q) AND HL(R)
          PUSH      DE                  ;SAVE y, HL HAS d (REMAINDER)
          LD        BC,4                ;d = d/4 + 1
          LD        DE,0
          CALL      DIVLW               ; (DE,HL)/BC TO DE(Q)
          INC       DE
          LD        HL,5                ;5*d - 3
          CALL      MULWW               ; DE*HL TO (DE,HL)
          LD        BC,3
          OR        A,A
          SBC       HL,BC
          JR        NC,JTOC2
          DEC       DE
JTOC2:    LD        BC,153              ;m = (5*d - 3)/153
          CALL      DIVLW               ; (DE,HL)/BC TO DE(Q) AND HL(R)
          PUSH      DE                  ;SAVE m, HL HAS d (REMAINDER)
          LD        BC,5                ;DAY = d/5 + 1
          LD        DE,0
          CALL      DIVLW               ; (DE,HL)/BC TO DE(Q)
          INC       DE
          LD        A,E
          LD        (DAY),A
          POP       HL                  ;m
          POP       DE                  ;y
          LD        A,L                 ;MONTH = m + 3
          ADD       A,3
          CP        A,12+1              ;IF MONTH .GT. 12,
          JR        C,JTOC3
          SUB       A,12                ;     MONTH = MONTH - 12
          INC       DE                  ;     y = y + 1
JTOC3:    LD        (MONTH),A
          LD        HL,YEAR0            ;YEAR = y + Base Year
          ADD       HL,DE
          LD        (YEAR),HL
          RET
```

(Continued on next page)

```
;MULTIPLY HL*DE, RETURN 32-BIT PRODUCT IN DE (HIGH) AND HL (LOW)

MULWW:   LD      B,H             ;MULTIPLICAND TO BC
         LD      C,L
         LD      HL,0            ;INITIALIZE PRODUCT
         LD      A,16            ;COUNT 16-BIT MULTIPLIER
MULWW1:  ADD     HL,HL           ;SHIFT DE,HL LEFT ONE
         EX      DE,HL
         ADC     HL,HL
         EX      DE,HL
         JR      NC,MULWW2       ;JUMP IF NO MULTIPLIER BIT
         ADD     HL,BC           ;ELSE ADD MULTIPLICAND TO RESULT
         JR      NC,MULWW2       ;IF LOW ORDER WORD OVERFLOWS,
         INC     DE              ;    CARRY INTO HIGH ORDER
MULWW2:  DEC     A               ;CONTINUE FOR 16 BITS
         JR      NZ,MULWW1
         RET


;DIVIDE (DE,HL) BY BC. QUOTIENT TO DE, REMAINDER TO HL.

DIVLW:   EX      DE,HL           ;HIGH ORDER TO HL, LOW TO DE
         LD      A,16            ;COUNT 16-BIT QUOTIENT
DIVLW1:  EX      DE,HL           ;SHIFT ONE BIT FROM DE TO HL
         ADD     HL,HL
         EX      DE,HL
         ADC     HL,HL
         SBC     HL,BC           ;WILL DIVISOR GO IN YET?
         JR      NC,DIVLW3
         ADD     HL,BC           ;NO - RESTORE HL
DIVLW2:  DEC     A               ;CONTINUE FOR 16 BITS
         JR      NZ,DIVLW1
         RET
DIVLW3:  INC     DE              ;INCREMENT QUOTIENT
         JR      DIVLW2          ;LEAVE RESIDUE IN HL, CONTINUE
```

End Listing

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

8088 Addressing Modes

The short comment on 8088 Base Relative Indexed Stack addressing and the table from Leo Scanlon's book, printed in this column a few months ago, drew a surprising number of letters. Albert Brunelli, of N. Chelmsford, Mass., practically wrote a whole tutorial; I found it so helpful that I am reprinting it verbatim below.

"I am writing in response to the 16-Bit Software Toolbox column of March 1983. The primary purpose of this letter is to explain some of the uses of the 8086/88's BP register. First I will comment on Leo Scanlon's listing on page 17 of the March 1983 issue.

"The subroutine will prove only that the DS register is not the default segment for BP-based memory references. The ASSUME statement does not change any register. It is merely a means of assuring the assembler that you know what you are doing when you make an anonymous reference (use a base or index as an offset without specifying a segment register). Mr. Scanlon appears to understand this when he initializes the DS register. However, he never initializes the SS register to the segment STACK which he would have to do to prove his case. If he were to change the SS to STACK, and if it weren't already pointing at STACK of course, he could never return from the subroutine. If the DL and DH registers hold 0FFh at the end of the routine, it is merely a coincidence. He would do better to push the data onto the stack and then retrieve it using the BP register. This technique will become clearer as we get further along.

"To get a good feel for the intended use of the BP register, one must understand the philosophy behind the use of the 8086 instruction set. As I understand it, Intel wanted to create a microprocessor with an instruction set which lent itself well to the implementation of high-level languages, specifically to Intel's systems language, PL/M.

"In PL/M and most other structured languages, most arguments are passed to procedures (functions, subroutines) on the stack. However, since the return address will be the lowest address on the stack, considerable manipulation must be done to access the arguments while maintaining the return address. Intel added two nice features to the instruction set to make argument retrieval easier. The BP register is one of these.

"Let's assume we wish to pass two arguments to the FAR subroutine SUB1. We might pass them as shown below:

```
        PUSH  CX
        PUSH  DX
        CALL  SUB1
```

"After the CALL instruction is executed, the stack will look like this:

```
low address   OLD IP   offset
              OLD CS   segment
              ARG2     from DX
high address  ARG1     from CX
```

"To retrieve the arguments and return properly, SUB1 might follow the procedure:

```
SUB1    PROC  FAR
        PUSH  BP
        MOV   BP,SP
        PUSH  DS     ;save registers
        PUSH  SI
        PUSH  AX
        ......       ;body of subroutine
        ......
DONE:   POP   AX
        POP   SI
        POP   DS
        POP   BP
        RET   4
SUB1    ENDP
```

"Moving SP to BP will allow us to index into the arguments passed on the stack using BP. The contents of the stack when we get to the body of SUB1 are shown as follows:

```
low address   OLD AX   ;"top" of stack
              OLD SI
              OLD DS
              OLD BP   ;BP points here
              OLD IP   ;BP+2
              OLD CS   ;BP+4
              ARG2     ;BP+6
high address  ARG1     ;BP+8
```

"We may thus retrieve the arguments with the instructions:

```
        MOV   AX,[BP+6]
                  ;fetch ARG2
        MOV   SI,[BP+8]
                  ;fetch ARG1
```

"Any reference using BP as a base will assume that it is in reference to the stack segment (with or without an ASSUME declaration).

"Now we have fetched the arguments without disturbing the return address. How will we clean up the stack so that the arguments will not clutter it up forever? Once again Intel comes to the rescue with the 'RET n' instruction. The 'n' part of the instruction tells the processor how many bytes to discard from the stack after it has taken off the return address. In other words, it will add 'n' to the stack pointer (SP) after fetching the return address.

"The final use to which I will put the BP register is one which I have found very helpful in memory-test programs which do not have access to the, as yet unverified, stack area of RAM. We may create synthetic subroutines which may be

```
            LEA   BX,TABLE_1          ;point to ROM table
            MOV   CX,NUM_ENTRIES      ;number of table entries
            LEA   BP,CONT1            ;BP becomes return addr
            JMP   SYN_SUB1
CONT1:      ......                    ;control returns here
            ......

SYN_SUB1;                             ;synthetic subroutine
            ......
            ......
            JMP   BP                  ;pseudo-return instr.
```

Figure 1.

'called' from anywhere within the same code segment. The idea is that the return address is placed in a register before jumping to the subroutine, which terminates with a jump register instruction such as shown in Figure 1, page 72.

"The LEA BP instruction will place the offset of the CONT1 label in the BP register. The BP register is the perfect one to use in a situation like this because we have no stack to which it may point. The JMP BP instruction may be thought of as roughly equivalent to the 8080's PCHL instruction, although it is more powerful since any general register may be used as the source."

Wishful Thinking Dept.

Jim Howell wrote to point out a truly glaring error, found on page 3-13 of the *iAPX-88 Book* (July 1981 edition), which readers of this column may find amusing or amazing. "16-bit operands are stored in memory with the most significant byte (MSB) first, followed by the least significant byte (LSB) in the next location." The picture set at the bottom of the page in the book repeats this error.

8088 Line Generator

Dan Rollins of Glendale, Calif. sent in an 8088 assembler version of the fast-vector algorithm featured in Dave Cortesi's column of December 1982 and February 1983. His comments follow:

"The routine is self-modifying and needs no external storage area. The code that calculates the coordinate pairs is blazing fast. It's a shame that the BIOS write-dot routine (which it calls) is so slow. I am using a version of this routine in an arcade game I am currently writing. All of the drawing and point testing in that game takes place in a buffer and I use a much faster PLOTDOT routine."

Listing 1 (page 75) contains the line generation subroutine proper, while Listing 2 (page 77) demonstrates how to call the subroutine from BASIC.

PC-TALK III

Andrew Fluegelman, author of the excellent PC-TALK communications program and originator of the "Freeware" concept, has announced a new version that has some significant enhancements. Version III has been rewritten so that all features operate at 1200 baud. The Key Directory has been expanded to 40 permanent strings, and the Dialing Directory now holds 60 entries and stores selective character stripping for each entry.

The Transmit and Receive routines now offer the options of line-paced transmission, sending of binary files, and the XMODEM error checking protocol. Program defaults can be reset interactively while the program is running. Other new bells and whistles include auto-redial, margin width alarm, and a screen dump to disk.

Updates to Version III for previous PC-TALK owners are only $10.00, while the suggested donation for new owners is $35.00. As before, everyone is encouraged to make copies of the program and distribute them as widely as possible for free; recipients of such copies are requested to mail in a donation if they appreciate the program and want to promote the development of more Freeware. PC-TALK is available from The Headlands Press, P.O. Box 862, Tiburon, California 94920.

(Listings begin on page 75)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 247**

The Line Generator Subroutine

Listing One (Text begins on page 72)

```
                              page ,132
                              title DRAWLINE.ASM       Dan Rollins

                              public drawline

                         ;8088 self-modifying program implements fast-vector algorithm
                         ; described by Michalsky, DDJ #74, 12/82
                         ; see also: FAST-LINE DRAWING TECHNIQUE, BYTE, Aug 81

                         ;routine expects to be called from BASIC via:

                         ;       CALL DRAWLINE(V%(0))

                         ;  where V%(0)   = X1    starting clm (0-319)
                         ;        V%(1)   = Y1    starting row (0-159)
                         ;        V%(2)   = X2    ending clm
                         ;        V%(3)   = Y2    ending row
                         ;        V%(4)   = color  (0,1,2,3)
                         ;        V%(5)   = length
                         ;                    0    = draw entire line
                         ;                    else = draw sub- or super-set of this vector

                         code    group   cseg
0000                     cseg    segment public 'code'
                                 assume  CS:cseg, DS:nothing, ES:nothing

                         ; make it easier to access variables and arguments
 =                       ARG1    equ     word ptr [BP+6]

 =                       X1      equ     word ptr [si]
 =                       Y1      equ     word ptr [si+2]
 =                       X2      equ     word ptr [si+4]
 =                       Y2      equ     word ptr [si+6]
 =                       COLOR   equ     byte ptr [si+8]
 =                       LEN     equ     word ptr [si+10]

                         ; these are values that will be inserted in the code
 = 0041                          INC_X   equ     41H
 = 0049                          DEC_X   equ     49H
 = 0042                          INC_Y   equ     42H
 = 004A                          DEC_Y   equ     4AH

                         ; these are the addresses where new code is overlayed
 =                       ADJ_LONG_AXIS    equ   byte ptr cs:[di]
 =                       ADJ_MASTER       equ   word ptr cs:[di+3]
 =                       TEST_MASTER      equ   word ptr cs:[di+7]
 =                       ALT_ADJ_MASTER   equ   word ptr cs:[di+13]
 =                       ADJ_SHRT_AXIS    equ   byte ptr cs:[di+15]

                               page
0000                     drawline proc    far
0000  55                        push     bp         ;always save
0001  8B EC                     mov      bp,sp

0003  8B 76 06                          mov     si,ARG1    ;si => address of X1 ;ie, V%(0)
```

(Continued on next page)

The Line Generator Subroutine

Listing One (Listing continued, text begins on page 72

```
0006  B3 41                    mov     bl,INC_X  ;assume Xstep = +1
0008  8B 44 04                         mov     ax,X2
000B  2B 04                    sub     ax,X1
000D  7D 04                    jge     dl1       ;if X1 <= X2 then no change

000F  B3 49                    mov     bl,DEC_X  ;Xstep = -1
0011  F7 D8                    neg     ax        ;Xdist = abs(Xdist)
0013                   dl1:
0013  8B C8                    mov     cx,ax     ;save Xdist

0015  B7 42                    mov     bh,INC_Y  ;assume Ystep = +1
0017  8B 44 06                         mov     ax,Y2
001A  2B 44 02                         sub     ax,Y1
001D  7D 04                    jge     dl2       ;if Y1 <= Y2 then no change

001F  B7 4A                    mov     bh,DEC_Y  ;Ystep = -1
0021  F7 D8                    neg     ax        ;Ydist = abs(Ydist)
0023                   dl2:
0023  8B D0                    mov     dx,ax                     ;save Ydist
0025  BF 005D R                mov     di,offset cs:modify_base ;point to the code
                                                                 ; to modify
0028  3B D1                    cmp     dx,cx                     ;determine longest axis
002A  7D 04                    jge     dl3                       ;Y is longer, so skip

002C  87 CA                    xchg    cx,dx              ;swap Xdist, Ydist
002E  86 DF                    xchg    bl,bh              ;swap INC/DEC X/Y values
0030                   dl3:                               ;modify:
0030  2E: 88 3D                mov     ADJ_LONG_AXIS,bh   ; the 1st INC/DEC code
0033  2E: 89 4D 03             mov     ADJ_MASTER,cx      ; main duty master adjustment
0037  D1 E9                    shr     cx,1               ;set up cycle tester
0039  2E: 89 4D 07             mov     TEST_MASTER,cx     ; test for cycling
003D  2E: 89 55 0D             mov     ALT_ADJ_MASTER,dx  ; alternate adjustment
0041  2E: 88 5D 0F             mov     ADJ_SHRT_AXIS,bl   ; alternate INC/DEC code

0045  8B FA                    mov     di,dx           ;DI is counter: long axis length
0047  83 7C 0A 00              cmp     LEN,0           ;if length arg > 0
004B  7E 03                    jle     dl4
004D  8B 7C 0A                         mov     di,LEN          ; then use it as counter
0050                   dl4:
0050  8B 0C                    mov     cx,X1
0052  8B 54 02                         mov     dx,Y1
0055  8A 44 08                         mov     al,COLOR
0058  33 DB                    xor     bx,bx           ;duty master starts = 0
                               page ;---------- top of vector plotting loop ---------
005A                   dl5:
005A  E8 0074 R                call    plotdot         ;plot a dot

005D                   modify_base     label   byte

005D  41                       inc     cx        ;INC/DEC CX/DX:  adjust long axis ptr
005E  81 C3 1111               add     bx,1111H  ;Xdist or Ydist: adjust duty master
0062  81 FB 1111               cmp     bx,1111H  ;Ydist or Xdist: check cycle position
0066  7E 05                    jle     dl6       ;skip if short axis is still ok
```

```
0068  81 EB 1111              sub     bx,1111H  ;Xdist or Ydist: adjust duty master
006C  42                      inc     dx        ;INC/DEC DX/CX:  adjust short axis ptr
006D                  dl6:
006D  4F                      dec     di        ;di is used as counter
006E  7D EA                   jge     dl5       ;do next dot if not finished
                              ;-------------------------------------------

0070  5D                      pop     bp        ;always restore
0071  CA 0002                 ret     2         ;back to BASIC, discard 1 arg <<EXIT<<
0074                  drawline endp

                      ;this routine plots the pixel at column CX  (0-319 or (0-639)
                      ;                                  row    DX  (0-199)
                      ;                                  color  AL  (0-3)  or (0-1)
0074                  plotdot proc    near
0074  50                      push    ax
0075  57                      push    di              ;BIOS destroys these registers

0076  B4 0C                   mov     ah,12           ;write_dot function
0078  CD 10                   int     10H             ;  video I/O call

007A  5F                      pop     di
007B  58                      pop     ax
007C  C3                      ret
007D                  plotdot endp

007D                  cseg    ends
```

End Listing One

Listing Two

```
    ' TESTER.BAS           Dan Rollins  02/02/83

    ' this compiled BASIC program tests the 8088-code DRAWLINE routine

     defint a-z
     dim v%(5)
     screen 1
10:
     input"routine (1 or 2)";r
     if r=2 then 100

20:
     input"color (0-3, -1 to quit)";v%(4)
     if v%(4) < 0 then 30

     input"x1,y1,x2,y2";v%(0),v%(1),v%(2),v%(3)

     v%(5)=0
     input"length (null = full line)";v%(5)

     call drawline(v%(0))

goto 20

30:
     screen 0,0,0,0 :width 80 :end
'-------------------------------------------
```

(Continued on next page)

Calling the Subroutine from BASIC

Listing Two (Listing continued, text begins on page 72)

```
' LASER BEAMS:  sample use of the length parameter

100:
    cls
    v%(0)=159 :v%(1)=(199)     'X1,Y1 = bottom center of screen
110:
    v%(2)=int(rnd*312)         'X2    = random column
    v%(3)=0                    'Y2    = top of screen

    v%(4)=3                    'color = white
    for length=10 to 190 step 10
       v%(5)=length            'set length
       call drawline(v%(0))    'draw partial line
    next
    v%(4)=0                    'color = black to erase
    for length=10 to 190 step 10
       v%(5)=length
       call drawline(v%(0))
    next

if inkey$="" then 110 else 30 'any key to exit
```

End Listing Two

DDJ T-Shirt

Dr. Dobb's Journal
COMPUTER
Calisthenics & Orthodontia

Don't underestimate the power of an original DDJ T-Shirt! Thousands of loyal DDJ readers requested them and after much anticipation they are finally available. These handsome royal blue T-Shirts are made of a fine polyester/ cotton blend. The cost of your priceless DDJ T-Shirt is ony $6.50. Use the attached coupon to order your T-Shirt today. Supply is limited, so don't delay. Sign up as a Donating Subscriber (see masthead) and receive a T-SHIRT FREE!

ORDER FORM

Make check or money order payable to:

Dr. Dobb's Journal
P.O. Box E
Menlo Park, CA 94025

Indicate Quantity: ____ S ____ M ____ L ____ XL

Total Quantity ________ x $6.50 = ________
(Sales tax is already included)

Add Shipping & Handling + 1.25

Add $1.00 for shipping
for each additional T-Shirt = ________

Total Amount ________

____ Check enclosed

____ Charge my: AmEx ____ Visa ____ MasterCard ____

Account No. ________

Exp. date ________ Signature ________

Name ________

Address ________

City/State ________ Zip ________

Phone ________

T3

CP/M EXCHANGE

by Robert Blum

Take a Look in the Public Domain...

Public domain software libraries continue to grow and the number of computerists involved in related activities is virtually exploding. The two predominate groups responsible for cataloging and distributing public domain software are the CP/M Users Group (CPMUG) and the Special Interest Group for Microcomputers (SIG/M). Collectively they offer over 140 volumes (almost 23 Mbytes) crammed full of programs. Many smaller groups specializing in specific programming languages and machines exist. Most have program libraries and are responsible for many of the software contributions of the larger groups.

CPMUG (1651 Third Avenue, New York, New York 10028) is the single largest distributor of CP/M public domain software, with 85 volumes in their library. No matter what your needs, from utilities to BASIC business software, it can be found in their catalog. At a nominal cost of $12 per volume, a better bargain can't be found. Unfortunately, one stumbling block exists. With each new computer comes a new disk format and stocking all of them is beyond the resources of most groups. In response to this need, a few commerical concerns are now offering copies of the original disks in many of the more popular disk formats.

What is desperately needed is some universal form of machine-readable media. Since none are currently available, two alternate approaches can be used. The least costly is "paperware." Even though the cost is low, many hours will be spent keying in the program and then finding the transcription errors. The second approach requires the purchase of a modem. With a 300 baud modem you will have ready access to a number of Remote CP/M (RCPM) systems across the country. Many of them offer complete libraries of public domain software in addition to an amazing number of other useful programs.

One often-overlooked point is the educational value of public domain software. Many fine books and reference documents are available to provide the prerequisite training on the computers instruction set and the CP/M interface. But none can rival the first-hand experience of stepping through a working program with a debugger. As if by magic, the mysteries are solved as you watch each instruction execute.

If a list of the foremost activists in public domain software were made, Ward Christensen would rank in the top ten. His involvement with microcomputers began with their inception. As librarian of CPMUG, he exemplifies dedication to the public domain scene. To ease the log jam of phone calls to his RCPM system he has added a second phone line as an "administrative" and general CP/M message system (messages deal with bug fixes, reviewing and cataloging of CPMUG disks, etc.). Use 312-849-1132 (and the new "M" command and "CPMUG" password) to download the current CPMUG catalog of programs (not the individual files), retrieve a copy of the user group contribution form, send in a contribution, etc. The original CBBS in Chicago is at 312-545-8086, which supports baud rates of 110-600. Press return several times for speed detect. When using these numbers, be patient – they get a lot of activity.

Periodically, I will be reviewing programs from the public domain that provide topical information. For the edification of those who are interested, I will offer complete program listings along with the review at my cost of printing and shipping.

...at SD-44

Displaying the diskette directory is probably one of the most-used functions of CP/M. Unfortunately, the built-in command DIR produces an unsorted, single-column display which is hardly satisfactory when several hundred file names are to be scanned. CPMUG volume 85 contains SD-44, which I believe is the latest version in a series of full-feathered directory programs. The display produced is sorted, and three columns wide. Each file's size is given and the total space remaining on the diskette is calculated. This may sound like a description of many others but SD-44 offers a number of other features. The eight options recognized are:

S – **system option**: includes files in the output.

F – **file option**: echoes the directory output to a disk file on the default drive, named "SD.DIR." If SD.DIR already exists, then the directory output will be *appended* to the end of the file. Otherwise, SD.DIR will be created as a new file.

U – **user option**: allows the specification of the user number for the directory in the form "U*n*" where the user number, *n*, must be greater than 0 and no larger than 15. This option allows no spaces between "U" and *n*, and cannot be used on pre-CP/M 2.0 systems.

A – **all users**: displays directories of all user areas starting at the user area specified in the U option. If the U option is omitted, displays start with the default user area and continue up to MAXUSR.

R – **reset option**: provides automatic resetting of the disk prior to performing directory seach, updating the allocation vector. Same as doing a Ctrl-C when changing disks, but handy if you didn't (such as when running a SUBMIT file). The RESFLG equate will force the R option unconditionally each time SD is run.

N – **no page option**: unconditionally disables the page pause option. SD will not put the page-pause prompt into the output file.

P – **printer option**: forces all console output to be echoed to the CP/M list device, with the most significant bit set to 0.

D – **all disk option**: allows SD to search all disk drives online starting with the disk drive specified or implied in the command line filename.

Outside of the obvious usefulness of SD-44, most of the techniques described in my series on BIOS internals can be found in this program. If you want a paperware copy of SD-44 and its documentation, send $5.00 to the address at the end of this column.

More on CP/M Plus

The real story behind CP/M Plus's features unfolds when the BDOS functions are examined. Rather than include all the BDOS functions, Table I (page 82) only lists the additions and changes.

Many of the BDOS functions are now MP/M compatible. This fact makes me wonder what DR has in mind for future releases; possibly adding concurrent operation to CP/M or providing a natural upgrade path to MP/M. Whatever the

case, to make full use of all the new features will require some changes to the application program. Whether CP/M 2.2 programs will run without change under CP/M Plus is a question that can best be answered by running it. Providing trick code has been kept to a minimum, the success ratio should be high.

Next month I will continue with memory mapping and more on CP/M Plus's internals. You can reach me directly if you want to order paperware for SD-44, or to discuss subjects from this column, at: Bob Blum, 5536 Colbert Trail, Norcross, Georgia 30092; (404) 449-8948.

DDJ

(Table I begins on page 82)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 249**

Table I.

CP/M Plus BDOS function addition and changes.

3 – *Auxiliary Input:* same as CP/M 2.2 reader input.

4 – *Auxiliary Output:* same as CP/M 2.2 punch output.

7 – *Auxiliary Input* status: new function call to test whether data is available at the auxiliary input device. Under CP/M 2.2 was *Get I/O Byte.*

8 – *Auxiliary Output* status: new function call to test whether data can be sent to the auxiliary device. Under CP/M 2.2 was *Set I/O Byte.*

38 – *Access Drive:* MP/M function call, provided for compatibility only.

39 – *Free Drive:* MP/M function call, provided for compatibility only.

41 – *Test and Write Record:* MP/M function call, provided for compatibility only.

42 – *Lock Record:* MP/M function call, provided for compatibility only.

43 – *Unlock Record:* MP/M function call, provided for compatibility only.

44 – *Set Multi-Sector Count:* allows from 1 to 128 128-byte sectors to be read or written in one operation. MP/M compatible.

45 – *Set BDOS Error Mode:* determines how errors are handled. MP/M compatible.

46 – *Get Disk Free Space:* determines how many free 128-byte sectors there are on the specified drive. MP/M compatible.

47 – *Chain to Program:* allows automatic chaining from one program to another. MP/M compatible.

48 – *Flush Buffers:* forces any remaining records marked for writing to be written. MP/M compatible.

49 – *Get/Set System Control Block:* allows the system control block to be changed.

50 – *Direct BIOS Calls:* allows direct BIOS calls to be made through BDOS functions. CP/M Plus no longer supports direct BIOS calls.

59 – *Load Overlay:* load a Resident System Extension (RSX) or overlay into memory.

60 – *Call Resident System Extension:* special function for loading RSXs only.

98 – *Free Blocks:* returns to free space any blocks that have been allocated, but not permanently recorded on disk.

99 – *Truncate File:* truncates the specified file to the indicated random record number.

100 – *Set Directory Label:* creates or updates the directory label which indicates what extended directory options are active. For example, perform access date and time stamping. MP/M compatible.

101 – *Return Directory Label Data:* returns the directory label data byte. MP/M compatible.

102 – *Read File Date Stamps and Password Mode:* returns password and time stamp mode for specified file. MP/M compatible.

103 – *Write File XFCB:* creates or updates the XFCB for a specified file. MP/M compatible.

104 – *Set Date and Time:* sets the internal date and time. MP/M compatible.

105 – *Get Date and Time:* returns the internal date and time. MP/M compatible.

106 – *Set Default Password:* allows setting of a file password before a file is accessed. MP/M compatible.

107 – *Return Serial Number:* returns the 6-byte CP/M Plus serial number.

108 – *Get/Set Program Return Code:* allows a program to set a termination code for access by other programs which may follow.

109 – *Get/Set Console Mode:* allows setting of control parameters for certain BDOS console functions.

110 – *Get/Set Output Delimiter:* allows the string delimiter used in function 9 to be set to another value.

111 – *Print Block:* print the block of data pointed to by a CCB on the console.

112 – *List Block:* print the block of data pointed to by a CCB on the system list device.

152 – *Parse Filename:* parse a filename and prepare a file control block for use.

OF INTEREST

by Michael Wiesenberg

Impressions of the West Coast Computer Faire

It seems *de rigueur* for computer columnists to chronicle their impressions of the Computer Faire, so here are mine. As you all know, the West Coast Computer Faire, run by Jim Warren (one of *DDJ's* first editors), happens each spring at San Francisco's Civic Auditorium and Brooks Hall, with symposia and workshops at the San Franciscan and Holiday Inn hotels. Many call the Faire the most important microcomputer conference of the year.

It was more crowded than last year. Also more hype. The most impressive display was that of **Perfect Software**, obviously designed with great thought by marketing folks. Some computer businesses are just beginning to realize that good products don't necessarily sell themselves; they have to be *sold.* The Perfect folks used the principle well, as they lured you up a carpeted ramp lit on either side by flashing bulbs synchronized to simulate a lighted path travelling into the darkened interior recesses wherein brilliant blue laser blasts pierced clouds of smoke. Displays and booths everywhere else in the Faire were out in the open and jammed up against the wares of other companies; but PS had, probably at considerable expense, bought a large corner nook and turned it into a mysterious subterranean cave. As you stepped off the ramp, your eyes slowly adjusted to the gloom and you found yourself on a railed platform from which two staircases descended into a maze of monitors on pedestals, all displaying Perfect Software products. Giant frameworks of steel girders supported laser-emitting devices that randomly shot blue pencils of light above the heads of the crowd, creating moire patterns in clouds of dry-ice steam. A disembodied, amplified, sepulchral voice, describing the virtues of various Perfect products, floated through and around the din of the crowd.

Elsewhere, **Texas Instruments** put on a robot show. A wheeled contraption carrying a TI 99/4A on a tray, having a TV lens for a face and a monitor mirroring all it saw where its mouth belonged, wandered through the crowd near the TI booth, trading wisecracks with a "real" person dressed up as Charlie Chaplin (and symbolizing, perhaps, the dominance of the TI machine over IBM?) and carrying on apparently intelligent conversations with bystanders. I am familiar enough with robotics to know that this "robot" must have been controlled at a distance by human beings. I looked around for, but could not find, some seemingly innocent person perpetually draining a coffee cup, but in actuality speaking into a microphone concealed therein.

Both software and hardware were on sale for greatly reduced prices. If you know precisely what you're looking for, computer fairs probably have the best bargains anywhere. But, *caveat emptor:* not all of the exhibitors will be back next year.

The machine that impressed me the most was the Dynalogic **Hyperion**, the truly portable, beautifully designed 16-bit system that I described last month. The version I saw, the Hyperion Plus, costs close to $5000, but you can get one without quite so many bells and whistles for under $3400.

The machine that may well start a new trend in quality low-cost computing is the **Humdinger**. This tiny Z80 color computer with CP/M (described in detail below) costs but $129 yet offers features found only in computers costing literally thousands more.

Ah, yes, then there's **Lisa**. Wonderful machine. Maybe the best user interface for a micro commercially available. The mouse is easy to use, moves quickly anywhere on the screen, and instantaneously displays information any way you want it, without having to touch the keyboard. But $10,000? If they want to compete in the same ballgame with the PC, Apple will have to price Lisa in the same ballpark.

Through all the chaos glided Jim Warren on roller skates, constantly coordinating his show through a walkie-talkie that rarely left his lips.

In addition to manufacturers with products to display, acquisition editors for major publishers kept an eye out for potential authors of promising material, and software makers were on the lookout for new programs. If you have a book or software to sell, you'll find buyers at the Faire.

On Sunday at 5 p.m., the sated crowds were quickly hustled out into some of the worst rain San Francisco had experienced in years.

I was suffering from sensory overload, but glad I had attended. The same thing happens when I spend more than a few hours at a first-rate art museum.

Hard PHD or Toaster for Your System

The **Computer Service Company** offers 16 Mb Winchesters for most systems for $2595, 8 Mb for $2195, and double 5Mb removable subsystems with two free cartridges for $2795. These are variously called **PHD 8x8, PHD 4x4, PHD 8x8KP** (for KayPro, for example), **PHD 8x8S** (S-100), all of which are 5¼-inch drives, and **Toaster** (two 3.9-inch drives). The systems include drive, parallel interface, Z80 adapter (or adapters for virtually any other computer), transportability between computers, power supply, diagnostics, format, sector sparing program, driver, six-month warranty, and (here's the best part) *free* installation by The Computer Service Company, FOB Mountain View, California. While I'm giving them a mention, The Computer Service Company also rents all kinds of computer equipment, from Osbornes at $5 a day, dot matrix printers at $2.50, floppy disk drives at $2.50, hard disks at $7.50, to monitors at $1.25 (add about one-third to include service), and they offer phone consultation on all forms of hardware and software maintenance and design for $35 per hour. **Reader Service No. 101.**

Graphics Like A Word Processor

Graphics Processing System, for 48K Apple II Plus, from **Stoneware**, manipulates and edits images like a word processor. It will also mix and change colors at will, edit or erase a portion of a picture merely by defining its boundaries, zoom or reduce any portion of a picture four or 16 times (with the stored image having a greater resolution than that on screen, reproducible depending on the capabilities of the printer or plotter), rotate images in two dimensions, duplicate images on screen and to and from disk, change proportionality of portions or all of images, overlay in separate colors, and access the 16K RAM card. The system is compatible with graphics tablets,

light pens, and various plotters and graphics printers. $179. **Reader Service No. 103.**

A Good Case

Comp Cases by the **Computer Case Company** make almost any small system portable, inexpensively. Literally a custom built suitcase that exactly fits your IBM PC, Apple, TRS-80, etc., various monitors, printers, and other accessories, the Comp Case is built from mahogany plywood, has padded handles, brass hardware, key locks, vinyl cover, triple-thick, saddle-stitched vinyl at the edges, rubber feet, custom interior foam padding, nylon velcro straps, and interlocking top and base. Once your system is in a Comp Case, you need never remove it, so you can leave cabling in place and easily set up the system. Comp Cases vary in price from less than $100 for many peripheral or accessory cases, to $109 for a housing for Apple II with one drive, to $129 for Apple II with two drives and a monitor or TRS-80 Model III, to $139 for Apple III with drives and printer. **Reader Service No. 105.**

A Real Humdinger

The **Humdinger** from **Venture Micro** is a Z80 color computer with a "real" keyboard, 4K RAM, 8K BASIC in ROM, eight-color video, four-voice sound generator, free game cartridge, RF modulator, and parallel, serial, cassette, cartridge, joystick, expansion, and EPROM interfaces, for $129. You can add 16K RAM for $39.95, 64K for $99, voice synthesizer for $69.95, disk controller for $75, 5¼-inch drive for $210, CP/M for $79, word processor for $45, Pascal for $59, an 8088 for $119 (you'll also need the 8088 BASIC ROM for $124.95), and an 8087 for $299, travel case, graphics table, 80-by-24 video, user-defined graphics, real time clock calendar, editor/assembler, various game cartridges and cassettes, extended BASIC, COBOL, Forth, C, Logo, and Pilot. Ten new cartridge and cassette programs per month are planned. **Reader Service No. 107.**

Dot Matrix Printer with Daisy Wheel Quality

The **Santec S700 Printer** from **Western Technology** offers letter quality by printing each line four times with minute advances between passes, at a throughput of 32 to 58 cps. Since it prints 960 dots per inch, you'll need a magnifying glass to distinguish this mode from a daisy wheel output. A correspondence mode makes two passes at each line, with quality better than that of the best dot matrix printers, at a throughput of 65 to 195 cps. And the draft quality mode is as good as the best dot matrix printers. Up to 12 fonts, ranging in size from 7 to 12 points, can be intermingled simultaneously, with no pause in printing. Formatting commands, inserted in the text, are performed by the printer, rather than the computer, with total user control over margins, spacing, tab settings, justification (incremental and proportional), centering, boldface, underlining, and graphics. This remarkable printer costs under $4000. Spellbinder, the CP/M word processing software from Lexisoft, has a special version configured for the Santec. **Reader Service No. 111.**

Two APLs

VIZ::APL from **EASI APL Systems** is virtual-memory APL for Z80 microcomputers, and some 16-bit microcomputers with a Z80 board. This full implementation of APL has an overlaid interpreter and virtual work area limited only by disk space, runs under CP/M, interfaces with high-resolution graphics, uses double-precision, floating-point arithmetic, and has dynamic symbol table allocation. I saw the language demonstrated on an Osborne I at the West Coast Computer Faire, and was impressed with its fast, direct-mode vector calculations and transformations. No price on this one, but use the Bingo Card for more information. **Reader Service No. 115.**

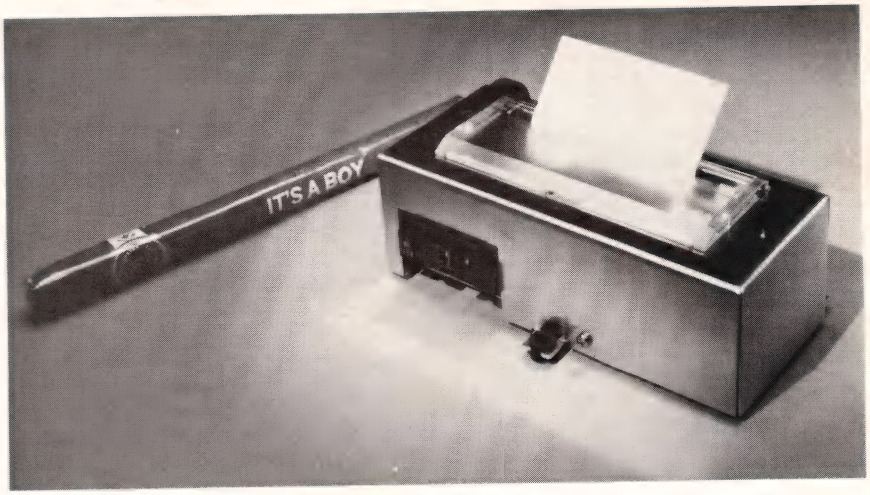
APL.68000 by the **Computer Company** is a full APL that runs under CP/M-68K, UNIX, and VersaDOS on the Sage Models 2 and 4, Forward Technology's Gateway Work Station microcomputers, the Pixel 100AP, the Corvus Concept, and "soon" on all 68000-based micros. **Reader Service No. 117.**

1/4

The **Forth-79 Version 2** compiler from **MicroMotion** for most Z80 CP/Ms 1.4 and 2.whatever, including Apple, comes with screen editor, macroassembler, string processing, three-bit arithmetic, floating point, 200 pages of tutorial and reference documentation, and hi-res for Apple and NorthStar, and costs from $99.95 (doesn't anything ever sell for an even $100?) to $139.95. **Reader Service No. 119.**

Wooden Clay?

Perhaps one of the better pieces of inexpensive software for the IBM PC seen at the West Coast Computer Faire was **Cosmic Nightware** from **Wood & Clay Hi-Tech Gameware.** "Earth explodes," they say (I saw it



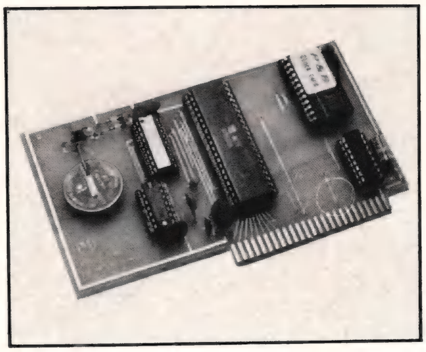
Hot Little Printer

The **EUY-3T** thermal printer from **Panasonic** is about the size of a good 5-cent cigar ("What this country needs is a good . . ." inexpensive printer), 4.69 by 1.79 by 2.64 inches. With dot-addressable graphics, 40 characters per line, battery power, bi-directional printing at 1.2 lines per second, and a weight of 14 ounces, the printer will cost under $100 in OEM quantities. **Reader Service No. 109.**

happen – wonderful graphics), "and you're the only one left! All the phantoms of the universe are released, and you have to fight enemies so real you won't believe it's a dream." The disk costs $32.50. If you want real arcade action on your PC, you'll want to order joystick controllers at $29.95 each from W&C. Other games available include **Falklands Fury**, with "all the action of real life warfare" (whoopee!, what fun!) and **Jungle Madness**, in which you "play Tarzan, swing from vines, and fight off savage pygmies," also $32.50 each. (California, add 6.5% sales tax.) **Reader Service No. 121.**

The Sorcerer Lives!

I often think my computer system is unusual and incompatible with anyone else's. When I say I have a Sorcerer, most people say, "A what?" So I'm always pleased to find someone trying to keep this wonderful Z80 machine alive. In this case, it's **ISIS** (the International Sorcerer Information Service), a "not-for-profit" monthly newsletter to give Sorcerer owners a means of exchanging information. Membership is $15 in Canadian funds in Canada, and US dollars elsewhere. Contact ISIS, c/o Maurice Dow, 84 Camberley Cres., Brampton, Ontario, Canada L6V 3L4; or use the reader service card. **Reader Service No. 123.**



Clock Your Apple

dat.a.clock, for Apple II, II Plus, and IIe, from **P & B Research Consultants** keeps time in date, month, and year, with a two- to three-year battery, has an externally accessible EPROM, and costs $55 in kit, or $85 assembled (plus $2 p&h). **Reader Service No. 113.**

Commodore Camaraderie

A VIC-20 users group is being formed by the **New York Amateur Computer Club**. Although the UG will meet in New York City, NYACC is soliciting members throughout the country to join "on a correspondence basis." Both types of VIC-20 users are sought, computer novices and the knowledgeable who bought the machine "just for the fun of it. What we would like to do is put these two types together and get some synergism going." If enough interest is generated, NYACC will plan to expand to other Commodore machines, like the 64 and the Pet. **Reader Service No. 125.**

Make a Beeline for the C Line

The C Line, a free public access electronic bulletin board for users of UNIX and C, has close to two megabytes of public domain C software just itching to be downloaded onto the computers of those with any standard terminal or computer with modem who dial (201) 625-1797, 8 pm to 9 am weekdays (Eastern time) and 24 hours weekends. The C Line, say sponsors **In-**

foPro Systems, publishers of *UNIQUE*, the UNIX industry newsletter, and Perchwell Corporation, a management research/consulting firm specializing in UNIX, "expects to receive several carriage returns initially" (ah, anthropomorphisms abound!), "and will automatically adjust itself to the transmission speed being used, from 110 to 710 baud." **Reader Service No. 129.**

Confer Intelligently

The National Conference on Artificial Intelligence, sponsored by the American Association for Artificial Intelligence, takes place August 22 to 26, 1983, at the Washington (D.C.) Hilton Hotel. **Reader Service No. 131.**

Keeping TABs

TAB Books' Spring/Summer catalog, with 65 new books and over 600 titles, includes these about computers:

Graphics Programs for the IBM PC (256 pages, $12.95 paper, $18.95 hardcover), *33 Games of Skill and Chance for the IBM PC* (256/$14.95/$21.95), and, possibly a good one for would-be hardware tech writers, *Beginner's Guide to Reading Schematics* (140/$8.95/ $13.95), all by Robert Traister.

100 Ready-to-Run Programs and Subroutines for the IBM PC, by Jeff Bretz and John Craig (320/$15.95/$22.95).

Programming Your Atari Computer, by Mark Thompson (280/$10.95/ $16.95).

Advanced Programming Techniques for Your Atari, including Graphics and Voice Programs, by Linda Schreiber (224/$13.95/$19.95).

25 Graphics Programs in Microsoft BASIC, by Timothy O'Malley (160/ $10.95/$17.95).

101 Projects for the Z-80, by Frank Tedeschi and Robert Colon (368/ $16.95/$23.95).

Troubleshooting and Repairing Personal Computers, by Art Margolis (320/ $13.95/$19.95).

Microcomputer-Controlled Toys and Games and How They Work, by Van Waterford (240/$9.95/$17.95).

Experiments in Four Dimensions, by David Heiserman (288/$12.95/$21.95).

Learning Simulation Techniques on a Microcomputer Playing Blackjack and Other Monte Carlo Games, by Pat Macaluso (154/$10.95/$16.95).

Basic BASIC-English Dictionary for the Apple, PET, and TRS-80, by Larry Noonan (154/$17.95, hard cover only).

The Sinclair ZX81, by Randle Hurley (182/$16.95, hard cover only).

Electronic Components Handbook for Circuit Designers, by R.H. Warring, (336/$13.95/$21.95).

Reader Service No. 127.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 251**

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems
ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems
ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

DDJ Begins Where The Others Leave Off

12 issues for just $25.

☐ **YES!** Please send me the next 12 issues of *Dr. Dobb's Journal* for $25.00. I save $5.00 off newsstand price.

☐ Please bill me ☐ I enclose check/money order

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

Card # ______________ Exp. Date ________

Signature ______________________

Name ______________________

Address ______________________

City ____________ State ______ Zip ______

Offer good in USA only. Foreign rates upon request.

Dr. Dobb's Journal
For Users of Small Computer Systems
A Publication of People's Computer Company

P.O. Box E
Menlo Park, CA 94025 (415) 323-3111

Dear Reader, **June 1983 #80**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps, or for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond. Simply fill it out and drop it in the mail. We take care of the rest.

Thanks for taking a few minutes to talk with us. – Ed.

Which articles or departments did you enjoy the most this month? Why? (Please indicate order of preference.)

____

____

Comments or suggestions:

____

____

____

____

Name ______________________

Address ______________________

____

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

EDITORIAL DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date.

Name ______________ Address ______________

City ______________ State ________ Zip ________ June 1983 #80

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
Articles: 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256

Profession:

1☐Programmer/Analyst
2☐Engineer/Scientist/Technician
3☐Business Owner/Manager
4☐Educator
5☐Consultant
6☐Professional (Law, Medicine, Accounting, etc.)
7☐Student
8☐Other ______________

Brand name of microcomputer you work with:

1. ______________
2. ______________
3. ______________

Purchase of magazine:

1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ______________

Comments:

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal
For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94025

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date.

Name ______________ Address ______________

City ______________ State ________ Zip ________ June 1983 #80

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
Articles: 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256

Profession:

1☐Programmer/Analyst
2☐Engineer/Scientist/Technician
3☐Business Owner/Manager
4☐Educator
5☐Consultant
6☐Professional (Law, Medicine, Accounting, etc.)
7☐Student
8☐Other ______________

Brand name of microcomputer you work with:

1. ______________
2. ______________
3. ______________

Purchase of magazine:

1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ______________

Comments:

Clinic

(Continued from page 12)

our CP/M 2 system had. So it was a disappointment to find that an assembly that ran in 25 seconds under CP/M 2 consumed 23 seconds under CP/M 3. A ten percent improvement didn't seem like much of a return on our investment in RAM cards. We set out to find out what the CP/M 3 BDOS was doing with all those buffers.

Inter-Bank Peeping

We wrote an experimental tool called SHOWBCB. It finds and dumps the two chains of the BCBs to the console. This is not easy, because the Disk Parameter Header, the chain anchors, and all the BCBs are in bank zero, while the program runs in bank one. It is made possible by the XMOVE and MOVE BIOS functions. MOVE is normally an intra-bank block copy function. If XMOVE is called first to inform MOVE of the source and target bank numbers, the next call on MOVE will transfer data from a source address in one bank to a target address in another. With them, our program could get copies of the control blocks into bank one so it could display them.

One problem slowed us down for a few hours. The program was written to play by the rules of CP/M 3, making its BIOS calls by way of BDOS function 50. It turns out, however, that the BDOS uses MOVE (and XMOVE?) in the course of executing function 50. So by the time we reached the BIOS for the MOVE function, the BDOS had already been there and thus had used up the parameters we had given to XMOVE. What we meant for an inter-bank move was thus magically transformed into an intra-bank move. We reverted to calling MOVE and XMOVE directly. This is possible provided that the BIOS routines and the calling program's stack are all in non-banked storage.

The Directory BCB Chain

Examine Figure 1 (page 91). It shows the BCB chain immediately after a cold start. The only disk I/O that has been done to this point is the load of CCP.COM and its load of SHOWBCB.COM. The lengthy display of the data BCBs has been trimmed to save space.

The first byte of a BCB is the number of the drive for which it holds a sector, and FFh indicates an unused buffer. Only one BCB is active in each chain. In the directory chain you can see that four BCBs have been used, but three of them have since been discarded. Old BCBs for track 2 sectors 1, 2, and 4 can be seen. Presumably sector 3 was read as well, but its BCB doesn't show.

Track 2 sector 4 was read again, and its BCB is still active. This BCB probably represents the I/O done to open SHOWBCB.COM. Thanks to its directory hashing table, the BDOS didn't have to scan the directory for SHOWBCB; it was able to read the one sector that contained its entry. However, it did have to read that sector because the residual BCB for the same sector had been discarded.

This seems reasonable on reflection. It's only by reading a directory sector and testing its contents against the hash table that the BDOS can detect a change of disks. If it didn't make that test, it would be as liable to trash a directory as MSDOS is. Extra directory I/O is a reasonable price to pay to avoid that danger to data integrity, so the fact that the BDOS seems to toss away its active directory BCBs shouldn't disturb us too much.

The Data BCB Chain

Now look at the chain of data BCBs. Two have been used, but only one remains active. The discarded one probably represents the last sector of CCP.COM; track 4 sector 6 is about the right location for it

on this disk. Of course, CCP.COM occupies 3.5 one-kilobyte sectors. Why does only one BCB appear? Probably because the BIOS's cold-start code uses a multi-record read to load the file. When the BDOS knows it will be transferring eight or more 128-byte units from adjacent disk locations to adjacent storage locations, it probably skips allocating a deblocking buffer and reads the sector directly into storage. The last sector of the file is not all used, so a BCB was allocated to hold it while the last few records were pulled out of it. The one active BCB probably represents the only sector of SHOWBCB.COM.

Examine the addresses in the "bcb" and "link" columns; you can see how the BDOS's LRU algorithm works. BEC3 is the last BCB assembled in the BIOS. When the system is loaded, it is at the end of the chain. Whenever the BDOS needs a BCB, it removes the last one from the end of the chain and inserts it at the head of the chain. In Figure 1, you can see that the BCB at BEC3 was moved from tail to head, and the same was done to the BCB that originally preceded it, BEB4. The original chain head, BAA9, has aged two positions. We can tell how many BCBs the BDOS has used by counting the BCBs that precede BAA9. This remains true only for 71 uses, when the chain wraps.

BCB Use for Input

Our next experiment was to run an assembly that did input but no output ("rmac showbcb $1brzszpz"). The assembler had to read a total of 29 sectors: ten sectors of SHOWBCB.ASM (twice each), and nine sectors in three library files. Figure 2 (page 91) shows the data BCBs afterward. The only one still active, BE4B, represents the load of SHOWBCB.COM. The two oldest, BEB4 and BEC3, are left over from Figure 1. The six in between are the total result of loading RMAC, reading the source file twice, and reading three libraries. The BCB at BEA5 probably represents the last sector of RMAC (it, like CCP.COM earlier, would have been loaded with a multi-record read). That leaves five BCBs to account for all 29 sectors of data.

Two things stand out in Figure 2. First, the BDOS does not use a new BCB for every new sector of data it reads. If it did, Figure 2 would be 24 BCBs longer than it is. Perhaps the BDOS assigns only one BCB to each open file (three libraries plus the source file twice equals five). Perhaps it reuses a BCB whenever its current sector is completely used up and the next sector is for the next adjacent record of the file. The second scheme would produce the same numbers in this case of all-sequential I/O, but would allow for the accumulation of more BCBs to a direct-access file.

Either way, the result is that for sequential input, the BDOS effectively

assigns a single BCB to each open file. Sequential data does not accumulate in the BCB chain. When a file is used twice in succession (as is the source file in an assembly), it will be read twice from disk. The only effect of the BCB chain is to eliminate the second read of the *last* sector. Now we can see why an assembly runs at essentially the same speed in CP/M 3 and CP/M 2.

The second thing that stands out in Figure 2 is that what little data was accumulated has since been discarded. The data residual from RMAC's execution was tossed out sometime between the end of RMAC and the start of SHOWBCB. The culprit had to be the CCP – it was the only program to run in that interval. We ran CCP.COM under SID to see what it would do.

What it did was issue BDOS request 98, Free Blocks. This BDOS function is supposed to release any disk blocks that were allocated to files that were never closed. It recovers disk space that might otherwise remain inaccessible until the disk was logged in again (possibly a long time in CP/M 3, since disks are only logged in when control-C is pressed). Exactly that problem occurs under MSDOS when the IBM Pascal compiler aborts, but under MSDOS you have to run CHKDSK to recover the space.

BDOS function 98 has a laudable goal, but it seems to have an unfortunate side effect. It not only cleans up the allocation vector, it also puts FFh into every active BCB! We proved that by killing the call on function 98 in CCP.COM (it's at location 458h) and re-running the experiments. With the modified CCP, the BCB chain after the assembly had eight active entries instead of eight dead ones and one live. However, this had little effect on performance, since the active entries represented only the last sectors of their respective files.

```
Directory BCB chain...

 bcb drive record wflg "00" track sector buffad bank link *

BA6D  00   000008  00   08   0002   0004  4000   00  BA7C
BA7C  FF   000008  00   08   0002   0004  4400   00  BA8B
BA8B  FF   000000  00   00   0002   0001  4800   00  BA9A
BA9A  FF   000018  00   07   0002   0002  4C00   00  BA31 *
BA31  FF   000000  00   00   0000   0000  3000   00  BA40
BA40  FF   000000  00   00   0000   0000  3400   00  BA4F
BA4F  FF   000000  00   00   0000   0000  3800   00  BA5E
BA5E  FF   000000  00   00   0000   0000  3C00   00  0000 #

Data BCB chain...

 bcb drive record wflg "00" track sector buffad bank link *

BEB4  00   000BF0  00   04   0031   0003  7800   03  BEC3
BEC3  FF   0000B8  00   00   0004   0006  7C00   03  BAA9 *
BAA9  FF   000000  00   00   0000   0000  5000   00  BAB8
BAB8  FF   000000  00   00   0000   0000  5400
BAC7  FF   000000  00   00   0000
BAD6  FF   000000  00   00                            03  BE69
BAE5  FF   000000                             6400   03  BE78
BAF4                           0000   0000  6800   03  BE87
                   00   00   0000   0000  6C00   03  BE96
      FF   000000  00   00   0000   0000  7000   03  BEA5
BEA5  FF   000000  00   00   0000   0000  7400   03  0000 #
```

Figure 1.

```
Data BCB chain...

 bcb drive record wflg "00" track sector buffad bank link *

BE4B  00   000BF0  00   04   0031   0003  5C00   03  BE69 *
BE69  FF   000C68  00   01   0033   0008  6400   03  BE5A *
BE5A  FF   000C60  00   07   0033   0005  6000   03  BE78 *
BE78  FF   000D08  00   04   0036   0004  6800   03  BE87
BE87  FF   000CE0  00   05   0035   0005  6C00   03  BE96
BE96  FF   000D30  00   03   0036   0003  7000   03  BEA5
BEA5  FF   0006E8  00   01   001D   0008  7400   03  BEB4
BEB4  FF   000BF0  00   04   0031   0003  7800   03  BEC3
BEC3  FF   0000B8  00   00   0004   0006  7C00   03  BAA9 *
BAA9  FF   000000  00   00   0000   0000  5000   00  BAB8
BAB8  FF   000000  00   00   0000   0000  5400
BAC7  FF   000000  00   00   0000
BAD6  FF   000000  00   00
BAE5  FF   000000
BAF4
```

Figure 2.

BCB Use for Output

We checked BCB use following an assembly that produced PRN, REL, and SYM files. We expected to see BCBs for at least the last sector of each output file. We did not. We don't know what space the BDOS is using to block the physical sectors of output files, but it isn't space taken from the BCB chain.

To make sure that we weren't seeing the result of some clever use of multi-record writes in RMAC, we ran a test using the Magic Wand PRINT program with disk output. This program has abysmal disk I/O logic; it alternates reading and writing of 128-byte units. Under CP/M 2, with a BIOS that had only a single sector buffer, PRINT caused a seek, read, seek, read, write sequence for every 128 bytes of throughput. Under our 2.2 BIOS with separate read and write sector buffers, it ran better. In fact, it ran better than under CP/M 3, where PRINT takes about 10% *longer* than under our improved 2.2 BIOS! And its output didn't leave any evidence in the BCB chain.

Here again, it seems that the CP/M 3 BDOS is throwing away an opportunity to speed up processing. All output goes straight to disk; if it then becomes input, it has to be read back again. The file that one program writes, another program is going to read. The editor's output is input to RMAC; RMAC's REL file is input to LINK; and its PRN file is input to TYPE or PIP. It's all very well to speed up direct access, but we should keep in mind that (1) the majority of CP/M commands read files sequentially, and (2) commands are used in sequence, with the output of one feeding the input of the next. That situation is exactly right for the LRU algorithm, but the BDOS isn't capitalizing on it.

Well, almost exactly right. If the BDOS buffered all the sectors it read, and also buffered all the sectors it wrote, some problems would arise. If an output file was larger than the total buffer pool, its last few sectors would displace its first few. The next command's input of the first few sectors of the file would kick out the next few, and so on. A ripple would run down the BCB chain, causing each saved sector to be discarded just a few sectors before it was needed as input.

But the BDOS does *not* buffer all the sectors it reads. If it kept the present logic for sequential reading but buffered all output sectors, then the ripple would never develop. One BCB would be used to read the leading sectors that had been displaced by later output. Eventually the input requests would work around to the sectors that had not been displaced. From there to the end of the file, the second command would operate at RAM speeds.

Summary

A sophisticated algorithm for sector buffering under CP/M would make special provision for COM files, encouraging the retention of commands in storage. It would probably use separate chains of BCBs for input and output, with some kind of heuristic to shuffle buffers from one chain to the other on the fly. In fact, CP/M 3 has enough resources to make it quite interesting as a case study in performance optimization. It could be the grist for a Ph.D. thesis in Comp. Sci., in fact.

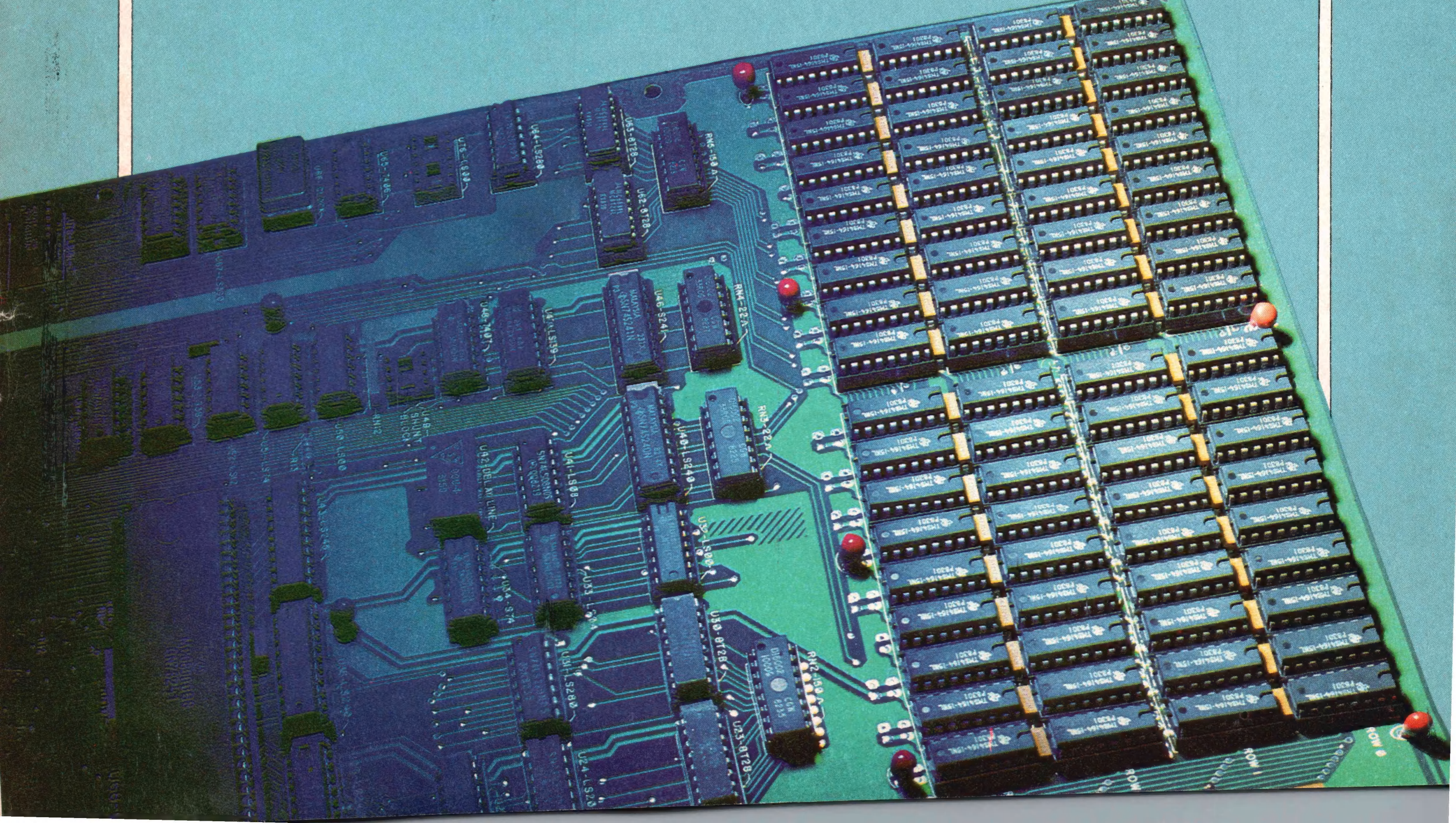
But the existing algorithm is not sophisticated and, while it may be effective for direct access, it is definitely not optimal for the kind of sequential access that is the bread and butter of any CP/M system. Until it is improved (CP/M 3.2?), we recommend that you not spend a lot of money on RAM cards when moving to CP/M Plus. One 48Kb system bank should provide as many sector buffers as the BDOS can use effectively.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 227**

ADVERTISERS INDEX

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, P.O. Box E, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025 • 415/323-3111



Which do you think is the more sophisticated computer?

Epson.

The big differences between the Epson HX-20 Notebook Computer (on the left) and the Apple Computer (on the right) are: 1) the HX-20 doesn't need a power cord, 2) the HX-20 weighs only about four pounds, and 3) the HX-20 costs a lot less money.

The Epson HX-20 Notebook Computer has a full-size keyboard, a built-in LCD screen, a built-in printer, 48K of combined RAM and ROM memory, and an internal power supply that will keep it running for over 50 hours. So you can do computing and word processing virtually anyplace you happen to be. Whereas, with the Apple Computer, you can only go as far as an extension cord will take you.

And on the HX-20, you get communications interfaces, upper and lower case letters, five program areas, a full 68 keys including an integrated numeric key pad, an internal clock/calendar, and the screen and printer. Standard. On the Apple, you pay something extra for each feature — if you can get them at all.

All of which makes the take-it-anywhere HX-20 perfect for business executives, salespeople, students, kids — anyone who's looking for an affordable, practical way into computing.

Portable. Powerful. Affordable. Sophisticated. The extraordinary HX-20 Notebook Computer. Find out just how extraordinary. Call (800) 421-5426, in California (213) 539-9140 for your nearest Epson computer dealer.

EPSON

EPSON AMERICA, INC.

Circle **no. 28** on reader service card.